cFS Test Framework (CTF) Tool

Software Design Document

Section 5.0

Contents

1 cFS Test Framework (CTF) Tool

- `CTF Software Requirements Specification (SRS)`
- `CTF Software Design Document (SDD)`

- CTF Software Test Procedures and Test Reports (STP)
- CTF Software User's Guide (SUG)
- CTF Test Instruction Reference
- **CTF Assumptions, Dependencies and Contraints** (See Sections 8 & 9 of the SUG.)
- **CTF Frequently Asked Questions** (See Section 11.1 of the SUG)

2 CCSDS Plugin

The CCSDS Plugin provides interfaces and utilities for CCSDS messages. It is responsible for parsing message structures and constructing messages with the correct header and payload formats.

Configuration

The CCSDS plugin reads some values from the `[ccsds]` section of CTF config file:

- **CCSDS_header_info_included:** Boolean indicating whether header info is included in the CCSDS exports
- **CCSDS_header_path:** The full file path of the module implementing CCSDS header types. The file does not need to be inside of the CTF directory. The CCSDS Plugin provides three header implementations: `ccsds_v1`, `ccsds_v2`, and `ccsds_gw`. To provide your own implementation, see Custom CCSDS Headers below.

ValidateCfsCcsdsData

Validates the format of CFS data types by sending one of each known command with an empty (all zeroes) payload.

- **target:** (string) The name of a registered CFS target. See CFS Plugin for registering targets.

Example:

```
{
    "instruction": "RegisterCfs",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1
},
{
    "instruction": "StartCfs",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1
},
{
    "instruction": "ValidateCfsCcsdsData",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1
}
```

Custom CCSDS Headers

The CCSDS Plugin provides default implementations of CCSDS message headers, and interfaces for implementing your own custom header types. Follow these steps to implement your own CCSDS header definitions, and refer to any of the provided implementations for further examples.

Create a new module

Create a new Python source file in the desired location. Import `ctypes` and declare classes for each of the primary header, a command packet, and a telemetry packet. These may extend the corresponding types provided by the CCSDS Plugin, or ultimately from `ctypes.Structure`. CCSDS headers typically extend from `ctypes.BigEndianStructure`.

Example:

```
import ctypes

 from plugins.ccsds_plugin.ccsds_primary_header import CcsdsPrimaryHeaderBase

 class MyPrimaryHeader(CcsdsPrimaryHeaderBase):
    pass

 class MyCmdPacket(ctypes.Structure):
    pass

 class MyTlmPacket(ctypes.Structure):
    pass
```

Define the fields and methods

Declare fields representing the bit structure of the headers. See `ctypes` documentation for details. Implement the necessary class methods to expose the field values. `CcsdsPacketInterface` provides an unimplemented interface for your convenience. You may also implement other structures and methods for internal use. At minimum, the following methods must be implemented:

- **Primary Header:** `get_msg_id()`, `is_command()`
- **Command Packet:** `get_msg_id()`, `get_function_code()`
- **Telemetry Packet:** `get_msg_id()`

Example:

```
class MyPrimaryHeader(ctypes.BigEndianStructure):
    _pack_ = 1
    _fields_ = [
        ("type", ctypes.c_uint16, 1),  # Packet type:      0 = TLM, 1 = CMD
        ("app_id", ctypes.c_uint16, 11),  # Application ID
        ("length", ctypes.c_uint16, 16)  # total packet length
    ]

     def is_command(self) -> int:
        return self.type

     def get_msg_id(self) -> int:
        return self.app_id
```

Export the types

Alias your types to `CcsdsPrimaryHeader`, `CcsdsCommand`, `CcsdsTelemetry` respectively for export. CTF will import and reference them by these names. In the CTF config file, set `ccsds:CCSDS_header_path` to the full path to your module.

Example:

```
CcsdsPrimaryHeader = MyPrimaryHeader
CcsdsCommand = MyCmdPacket
CcsdsTelemetry = MyTlmPacket
```

Test

Use the `ValidateCfsCcsdsData` in a test script to validate the header definitions as shown above. If the implementing module contains errors or does not meet the minimum requirements of CTF, `RegisterCfs` will fail and print an error message. Check CFS output to ensure that it recognized each of the messages and MIDs.

3 CFS Plugin

The CFS Plugin provides CFS command/telemetry support for CTF. The following test instructions are available.

Configuration

The CFS plugin draws many default values from the CTF config file. The section `[cfs]` defines defaults for all CFS targets and is always required.

If multiple CFS targets are to be registered, for each target name, the plugin will load values from a correspondingly named section.

If no targets are explicitly registered by name by the time `StartCfs` is first executed, the plugin will automatically configure targets for each config section beginning with `cfs_`. If no such sections are found, the plugin will configure a single target using the `[cfs]` config section. Note that if the `cfs_protocol` field is not found in the `cfs` section, a local target will be registered.

The precedence of values is first the named config section, if any, and then the `[cfs]` config section. A target cannot be registered, explicitly nor automatically, without a correspondingly named config section.

Example

‘‘‘

Base settings for cfs

[cfs] ...

Override settings for cfs_LX1

[cfs_LX1] ...

Override settings for cfs_workstation

[cfs_workstation] ...

Override settings for remote_cfs

[remote_cfs] "'

In this case, the test script may explicitly register target(s) named any of `cfs`, `cfs_LX1`, `cfs_workstation`, or `remote_cfs`. If no targets are explicitly registered, the plugin will configure targets `cfs_LX1` and `cfs_workstation` automatically because they match the naming convention, but not `remote_cfs`: it must be explicitly registered. The following examples assume that a target `cfs_workstation` has been registered.

A number of configuration fields relate to the CCDD files and formats, which will likely vary by project:

- `cfs:CCSDS_data_dir` provides the path to the directory containing CCDD JSON files for this target.
- `cfs:CCSDS_target` provides the target name found in the CCDD JSON files to identify MID values for this target.
- `cfs:log_ccsds_imports` will log details of CCDD JSON parsing for this target.
- `cfs:evs_event_mid_name` provides the name of the EVS event MID which must match the name given in CCDD JSON.
- `ccsds:CCSDS_header_path` provides the path to the module implementing CCSDS header definitions for all targets.

Test Script Considerations

CTF supports resolving macros from the ccsds_data_dir and replacing macros in the test script with the actual "c_value". Ensure a # precedes the macro in the test script in order for CTF to do macro replacement.

Example

```
{
                    "instruction": "CheckTlmValue",
                    "data": {
                        "mid": "CFE_EVS_HK_TLM_MID",
                        "args": [
                            {
                                "variable": "Payload.AppData[#MY_APPDATA_INDEX].AppID",
                                "value": [
                                    "0"
                                ],
                                "compare": "=="
                            }
                        ],
                        "target": ""
                    },
                    "wait": 1
                }
```

RegisterCfs

Declares a CFS target to be loaded according to the config file section of the same name. Any fields not provided in the named section will fall back to the CFS default values. The named section must contain, at minimum, a value for `cfs_protocol`, and may override any value specified in the `[cfs]` section.

- **target:** (string) A unique name to identify the target in later instructions. The name must match a section name in the config file.

Example:

```
{
    "instruction": "RegisterCfs",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1,
}
```

Config:

```
[cfs_workstation]
cfs_protocol="local"
...
```

BuildCfs

Builds a CFS target.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.

Example:

```
{
    "instruction": "BuildCfs",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1,
}
```

StartCfs

Starts a CFS target.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **run_args:** (Optional) Specify command line arguments to start CFS with. The value is appended to the `cfs_-run_args` defined in the configuration INI file.

Example:

```
{
    "instruction": "StartCfs",
    "data": {
        "target": "cfs_workstation",
        "run_args": "-R PO"
    },
    "wait": 1
}
```

EnableCfsOutput

Enables CFS output. No parameters.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered target.

Example:

```
{
    "instruction": "EnableCfsOutput",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1,
}
```

SendCfsCommand & SendCfsCommandWithPayloadLength

Constructs and sends a command message to CFS with the specified MID, command code, and payload arguments. **Note:** `SendCfsCommandWithPayloadLength` was named `SendInvalidLengthCfsCommand` prior to CT-F v1.4

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **mid:** The message ID of the command (i.e. "BEX_CMD_MID") (string)
- **cc:** The command code for the command (i.e. "BEX_NOOP_CC") (string)
- **payload_length:** (Optional) The size of the payload in bytes for a manually sized command. Do not specify for valid fixed-size commands. The actual length of the message will include the header size.
- **args:** An object where the key is the argument name, and the value is the argument value. Because `args` is a dictionary, the order does not matter. (i.e. `{"field_b": 1, "field_a": 0}` is equivalent to `{"field_a": 0, "field_b": 1}`)
- **header:** (Optional) An object where the key is the header field name, and the value is the field value. This object is passed into to the `CcsdsCommand` type (as determined by the config field `‘ccsds:CCSDS_header_-path‘`) and is not handled by CTF directly. It is made available for custom CCSDS header implementations to allow specification of the packet header.

Example:

```
{
    "instruction":"SendCfsCommand",
    "data":{
        "target": "cfs_workstation",
        "mid":"TO_CMD_MID",
        "cc": TO_ENABLE_OUTPUT,
        "args": {
            "cDestIp": "127.0.0.1",
            "usDestPort": "5011"
            }
        },
    "wait": 1
}
```

SendCfsCommandWithRawPayload

Constructs and sends a command message to CFS with the specified MID, command code, and payload bytes. The payload type for this command must be a byte array.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **mid:** The message ID of the command (i.e. "BEX_CMD_MID") (string)
- **cc:** The command code for the command (i.e. "BEX_NOOP_CC") (string)
- **hex_buffer:** A hexadecimal string representing the command payload. The string must be an even length (2 characters per byte) no larger than the command payload size and contain only hex numerals 0-F.
- **header:** (Optional) An object where the key is the header field name, and the value is the field value. This object is passed into to the `CcsdsCommand` type (as determined by the config field `‘ccsds:CCSDS_header_path‘`) and is not handled by CTF directly. It is made available for custom CCSDS header implementations to allow specification of the packet header.

Example:

```
{
    "instruction": "SendCfsCommandWithRawPayload",
    "data":{
        "target": "cfs_workstation",
        "mid": "DUMMY_IO_CMD_MID",
        "cc": "DUMMY_IO_RAW_BYTE_CC",
        "hex_buffer": "0123456789ABCDEF"
    },
    "wait": 1
}
```

CheckEvent

Checks that an event message matching the given parameters has been received from the CFS target. **Note:** This instruction's syntax changed in CTF v1.4

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **args**: an array of argument objects that describe the events to be checked. Multiple arguments can be listed here to check multiple events at once.
  - **app_name**: The app that sent the event message.
  - **event_id**: The Event ID, taken from an EVS enum, to represent the criticality level of a message. 13 is information, 14 is error, and anything else should be updated into this wiki as you find it.
  - **event_str**: (Optional) The expected message of the event. If blank, the event_str field is not verified.
  - **is_regex**: (Optional) True if `event_str` is to be used for a regex match instead of string comparison

  - **event_str_args**: (optional) arguments that will be inserted into `event_str`, similar to printf() functions

Example:

```
{
    "instruction":"CheckEvent",
    "data":{
        "target": "cfs_workstation",
        "args": [
          {
            "app_name":"BEX",
            "event_id":13,
            "event_str":"Processed MODE(%d) Command Successfully Received",
            "is_regex": false,
            "event_str_args":"(1,)"
          },
          {
            "app_name": "TO",
            "event_id": "3",
            "event_str": "TO - ENABLE_OUTPUT cmd succesful for routeMask:0x00000001"
           },
        ]
    },
    "wait": 1
}
```

CheckNoEvent

Checks that an event message matching the given parameters is no longer valid in received messages from the CFS target. **Note:** This instruction's syntax changed in CTF v1.4

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **args**: an array of argument objects that describe the events to be checked. Multiple arguments can be listed here to check multiple events at once.
  - **app_name**: The app that sent the event message.
  - **event_id**: The Event ID, taken from an EVS enum, to represent the criticality level of a message. 13 is information, 14 is error, and anything else should be updated into this wiki as you find it.
  - **event_str**: (Optional) The expected message of the event. If blank, the event_str field is not verified.
  - **is_regex**: (Optional) True if `event_str` is to be used for a regex match instead of string comparison
  - **event_str_args**: (optional) arguments that will be inserted into `event_str`, similar to printf() functions

Example:

```
{
  "instruction": "CheckNoEvent",
  "data": {
          "target": "cfs_workstation",
          "args": [
            {
              "app_name": "TO",
              "event_id": "3",
              "event_str": "TO - ENABLE_OUTPUT cmd succesful for routeMask:0x00000001",
              "event_str_args": "",
             }
           ]
   },
  "wait": 4,
  "description": "ENABLE_OUTPUT cmd message is no longer valid in received messages"
}
```

CheckTlmValue

Checks that a telemetry message matching the given parameters has been received from the CFS target.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **mid**: The telemetry message ID to check.
- **args**: an array of argument objects that describe the values to be checked. Multiple arguments can be listed here to check multiple attributes of a given packet at once.
  - **compare**: How to compare the telemetry value with the test value. Must be one of: ==, <=, <, >, >=, !=, `streq` (string equal), `strneq` (string not equal), `regex` (any regex match on a string).
  - **variable**: The attribute in the telemetry packet to check against.
  - **expected_mid** (optional): The telemetry message ID where the expected value can be found. Only needed if the check will be performed between two variables. This must match a name that was defined for this MID in the CCDD. (string)
  - **value**: The value to compare against. (number, string, bool) Note that the single value must be contained in a list: ∗∗"value":[0]∗∗, not ∗∗"value":0∗∗. Also if the command is called within a function and the value is a function parameter, put the parameter name as a string: ∗∗"value":["myParamName"]∗∗. If **expected_mid** is set, this field should contain the variable path to be checked.
  - **tolerance**: floating point tolerance.
  - **tolerance_plus**/**tolerance_minus**: non-symmetric floating point tolerance.

Example:

```
{
    "instruction": "CheckTlmValue",
    "data": {
        "target": "cfs_workstation",
        "mid": "TO_HK_TLM_MID",
        "args": [
            {
                "compare": "==",
                "variable": "usCmdErrCnt",
                "value": [
                    1
                ]
            },
            {
                "compare": "==",
                "variable": "usCmdCnt",
                "value": [
                    3.05
                ],
                "tolerance_plus": 0.1,
                "tolerance_minus": 0.1
            }
        ]
    },
    "wait": 1
}
```

CheckTlmPacket

Checks that a telemetry message with the given MID has been received from the CFS target. This is equivalent to `CheckTlmValue` without comparing args.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **mid**: The telemetry message ID to check.

Example:

```
{
    "instruction": "CheckTlmPacket",
    "data": {
        "target": "cfs_workstation",
        "mid": "TO_HK_TLM_MID"
    },
    "wait": 1
}
```

CheckNoTlmPacket

Checks that a telemetry message with the given MID is no longer valid in received messages from the CFS target.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **mid**: The telemetry message ID to check.

Example:

```
{
    "instruction": "CheckNoTlmPacket",
    "data": {
        "target": "cfs_workstation",
        "mid": "TO_HK_TLM_MID"
    },
    "wait": 1
}
```

CheckTlmContinuous

Similar to `CheckTlmValue` except the check is performed each time telemetry is received, until the test ends or the check is removed by `RemoveCheckTlmContinuous`.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **verification_id**: A unique string to identify this check within the test.
- **mid**: The telemetry message ID to check.
- **args**: an array of argument objects that describe the values to be checked. Multiple arguments can be listed here to check multiple attributes of a given packet at once.
    - **compare**: How to compare the telemetry value with the test value. Must be one of: ==, <=, <, >, >=, !=, `streq` (string equal), `strneq` (string not equal), `regex` (any regex match on a string).
    - **variable**: The attribute in the telemetry packet to check against.

- **expected_mid** (optional): The telemetry message ID where the expected value can be found. Only needed if the check will be performed between two variables. This must match a name that was defined for this MID in the CCDD. (string)
- **value**: The value to compare against. (number, string, bool) Note that the single value must be contained in a list: ∗∗"value":[0]∗∗, not ∗∗"value":0∗∗. Also if the command is called within a function and the value is a function parameter, put the parameter name as a string: ∗∗"value":["myParamName"]∗∗. If **expected_mid** is set, this field should contain the variable path to be checked.
- **tolerance**: floating point tolerance.
- **tolerance_plus/tolerance_minus**: non-symmetric floating point tolerance.

Example:

```
{
    "instruction": "CheckTlmContinuous",
    "data": {
        "target": "cfs_workstation",
        "verification_id": "TO_no_errors",
        "mid": "TO_HK_TLM_MID",
        "args": [
            {
                "compare": "==",
                "variable": "usCmdErrCnt",
                "value": [
                    0
                ]
            }
        ]
    },
    "wait": 1
}
```

RemoveCheckTlmContinuous

Cancels a continuous telemetry check by ID so that it is no longer performed.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets. Note that the target must match with the target specified in `CheckTlmContinuous`, otherwise the instruction will fail as it tries to remove the ID on the wrong target.
- **verification_id:** The ID of a check previously added by `CheckTlmContinuous`

Example:

```
{
    "instruction": "RemoveCheckTlmContinuous",
    "data": {
        "target": "cfs_workstation",
        "verification_id": "TO_no_errors"
    },
    "wait": 1,
}
```

ArchiveCfsFiles

Copies files from a directory that have been modified during the current test run into the test run's log directory.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.
- **source_path:** A directory path, absolute or relative to the location of CTF, from which to copy files

Example:

```
{
    "instruction": "ArchiveCfsFiles",
    "data":{
        "target": "cfs_workstation",
        "source_path": "../../build/exe/lx1/cf/"
    },
    "wait": 1
}
```

ShutdownCfs

Shuts down a CFS target explicitly within the test script. Note, the CFS plugin will automatically shutdown all CFS targets on test completion.

- **target:** (Optional) A previously registered target name. If no name is given, applies to all registered targets.

Example:

```
{
    "instruction": "ShutdownCfs",
    "data": {
        "target": "cfs_workstation"
    },
    "wait": 1,
}
```

4 Control Flow Plugin

The Control-Flow Plugin provides the functionality of CTF control flow statement at instruction level. It includes looping and conditional statements.

BeginLoop

Create a loop entry point. The loop is identified by a unique label. The BeginLoop must be in pairs with EndLoop instruction. The loop condition is defined in parameter "conditions" as a list of variables and the associated comparison operations. The condition is True, only if all comparison operations are True.

- **label**: a user defined label (example: "LOOP_1")
- **conditions**: a list of comparison conditions. Each includes "name", "operator" and "value".
- **variable**: either a variable defined by user or a variable from telemetry.
- **compare**: the operator applied to variable, including "<", "<=",">", ">=", "==", "!=" (example: "<")
- **value**: numerical number (example: 20)

Example:

```
{
    "instruction":"BeginLoop",
    "data": {
       "label": "LOOP_1",
       "conditions": [
                 {"variable": "my_var", "compare": "<", "value": 20},
                 {"variable": "tlm_usCmdCnt", "compare": "<", "value": 7}
       ]
      }
 }
```

EndLoop

Create a loop exit point. It must match a BeginLoop instruction with the same label. If the looping condition in BeginLoop is False, the control flow jumps to the corresponding EndLoop instruction, and exits the loop.

- **label**: a user defined label (example: "LOOP_1")

Example:

```
{
     "instruction": "EndLoop",
     "data": {
        "label": "LOOP_1"
     }
}
```

IfCondition

Create an entry point for if conditional branch block. It is identified by a unique label per test script. The IfCondition must be in pairs with EndCondition instruction. ElseCondition instruction is optional. The condition is defined in parameter "conditions" as a list of variables and the associated comparison operations. The condition is True, only if all comparison operations are True.

- **label**: a user defined label (example: "If_Label_1")
- **conditions**: a list of comparison conditions. Each includes "name", "operator" and "value".
- **variable**: either a variable defined by user or a variable from telemetry.
- **compare**: the operator applied to variable, including "<", "<=",">", ">=", "==", "!=" (example: "<")
- **value**: numerical number (example: 7)

Example:

```
{
    "instruction":"IfCondition",
    "data": {
       "label": "If_Label_1",
       "conditions": [
                 {"variable": "my_var", "compare": "<", "value": 10},
                 {"variable": "tlm_usCmdCnt", "compare": "<", "value": 7}
       ]
      }
}
```

ElseCondition

Create an entry point for else conditional branch block. The instruction is optional. But if defined, it must match a IfCondition and a EndCondition instruction with the same label. If the condition of IfCondition instruction is False, the control flow skips the 'if' branch block, only executes the 'else' branch block. If ElseCondition instruction is not defined, the control flow jumps to the end of conditional branch block defined by a EndCondition instruction.

- **label**: a user defined label (example: "If_Label_1")

Example:

```
{
     "instruction": "ElseCondition",
     "data": {
        "label": "If_Label_1"
     }
}
```

EndCondition

Create an exit point for if conditional branch block. It must match a IfCondition instruction with the same label. When the control flow reaches EndCondition instruction, it exits the conditional branch block.

- **label**: a user defined label (example: "If_Label_1")

Example:

```
{
     "instruction": "EndCondition",
     "data": {
        "label": "If_Label_1"
     }
}
```

5 Example Plugin

The Example Plugin shows a simple CTF plugin that can perform a single test instruction and a single verification instruction.

TestCommand

Simply logs that the test command was executed with the provided arguments.

- **arg1**: any value (example: "Hello")
- **arg2**: any value (example: "World")

Example:

```
{
    "instruction":"TestCommand",
    "data":{
        "arg1": "foo",
        "arg2": 42
    }
 }
```

TestVerifyCommand

Increments the plugin's example_counter value and checks if it is greater than `5`. CTF will poll run that instructions until the verification is successful, or a timeout occurs.

Example:

```
{
    "instruction":"TestVerifyCommand",
    "data":{}
}
```

TestSharedLibraryCommand

Uses libc to get the system time and log it to system output. Verifies that the expected number of bytes were printed. Example:

```
{
    "instruction":"TestSharedLibraryCommand",
    "data":{}
}
```

6 SSH Plugin

The SSH Plugin provides remote and local shell command execution capability for CTF. The following test instructions are available.

SSH_RegisterTarget

Declares a target host by name. This command must be run before any other commands given the same name will work. Command may be used multiple times to declare any number of targets. If not used, the plugin will assume that all commands are intended for the same target as defined in `SSH_InitSSH`

- **data**: an object where the key is the argument name, and the value is the argument value
  - **name**: An arbitrary, unique name to identify the target in subsequent commands. Does not need be the actual hostname of the target. Name is optional in all other commands, but if not provided all such commands will share a single connection.

Example:

```
    {
        "instruction": "SSH_RegisterTarget",
        "wait": 1,
        "data": {
            "name": "workstation"
         }
    }
```

SSH_InitSSH

Establishes an SSH connection with a target host. This command must be run before other remote commands will work. Command may be used multiple times with the same name to connect to different remote hosts in succession, or be used with different names to maintain concurrent connections to multiple hosts.

- **data**: an object where the key is the argument name, and the value is the argument value

  - **host**: hostname or IP to connect to, which may include the username and/or port.
  - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
  - **user**: User name for the connection. Do not use if you specified the user in `host`. (Optional)
  - **port**: Port number for the connection. Do not use if you specified the port in `host`. (Optional)
  - **gateway**: SSH gateway command string to proxy the connection to `host` (Optional)
  - **ssh_config_path**: Path to an ssh config file which may contain host definitions or additional parameters. If not specfied, `~/.ssh/config` will be assumed. (Optional)
  - **args**: Additional SSH connection options, as needed. See `Paramiko API docs` for relevant values. (Optional)

Note - CTF does not currently handle password entry/storage. Follow the tutorial `here` to set up SSH key authorization Example:

```
{
    "instruction": "SSH_InitSSH",
    "wait": 1,
    "data": {
        "name": "workstation",
        "host": "123.123.123.1",
        "user": "lander_demo"
        "port": 22
        "gateway": "ssh -W %h:%p myproxy"
        "ssh_config_path": "./ssh/config"
     }
}
```

SSH_RunRemoteCommand

Executes a command on the remote host. **ExecutionInitSSH** must be called first to establish an SSH connection.

- **data**: an object where the key is the argument name, and the value is the argument value
  - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
  - **command**: The shell command to be executed. Can contain multiple commands separated with `;`

Example:

```
{
    "instruction": "SSH_RunRemoteCommand",
    "wait": 1,
    "data": {
        "name": "workstation",
        "host": "cd lander_fsw_ctf/;rm -rf build; make; make install;"
     }
}
```

SSH_RunLocalCommand

Executes a command on the local host (the machine running CTF), regardless of the target. This is different from calling `SSH_RunRemoteCommand` targeting localhost, as it is invoked directly by the current process rather than passed via SSH.

- **data**: an object where the key is the argument name, and the value is the argument value

  - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
  - **command**: The shell command to be executed. Can contain multiple commands separated with `;`

Example:

```
{
    "instruction": "SSH_RunLocalCommand",
    "wait": 1,
    "data": {
        "name": "workstation",
        "host": "cd lander_fsw_ctf/;rm -rf build; make; make install;"
     }
}
```

SSH_CheckOutput

Compares the output of the most recently executed command. **ExecutionRunRemoteCommand** or **ExecutionRunLocalCommand** must be called first.

- **data**: an object where the key is the argument name, and the value is the argument value
  - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
  - **output_contains** (optional): A substring that must be contained in stdout. (Example: "PASS")
  - **output_does_not_contain** (optional): A substring that should *not* be contained in stdout. (Example: "FAIL")
  - **exit_code** (optional, default = 0): The expected exit code after the shell command is executed.

Example:

```
{
    "instruction": "SSH_CheckOutput",
    "wait": 0,
    "data": {
        "name": "workstation",
        "output_contains": "Built target mission-install",
        "output_does_not_contain": "Error",
        "exit_code": 0
    }
}
```

SSH_PutFile

Copies a path (file or directory) from the local filesystem to the remote host via rsync. Relative or absolute paths are allowed, but do not use $\sim$. Strings are passed directly to rsync, so the same rules apply regarding paths, patterns, etc.

- **data**: an object where the key is the argument name, and the value is the argument value
  - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
  - **local_path**: The path to the local file or directory to be copied
  - **remote_path**: The path to where the file or directory is to be copied. For remote hosts use the SSH syntax _user:path_.
  - **args**: An object that describes optional parameters for the transfer
    - **delete**: A boolean corresponding to `rsync`'s `--delete` option. If true, `rsync` will remove remote files that no longer exist locally. Defaults to false.

        * **exclude**: A string or array of strings corresponding to `rsync`'s `--exclude` option. Defaults to None.

Example:

```
{
    "instruction": "SSH_PutFile",
    "wait": 0,
    "data": {
        "name": "workstation",
        "local_path": "./cfs",
        "remote_path": "/tmp/workspace/cfs",
        "args": {
            "delete": true,
            "exclude": "*.git"
        }
    }
}
```

SSH_GetFile

Copies a path (file or directory) from the remote host to the local filesystem via rsync. Relative or absolute paths are allowed, but do not use $\sim$. Strings are passed directly to rsync, so the same rules apply regarding paths, patterns, etc.

- **data**: an object where the key is the argument name, and the value is the argument value
    - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
    - **remote_path**: The path to the source file or directory to be copied. For remote hosts use the SSH syntax _user:path_.
    - **local_path**: The local path to where the file or directory is to be copied
    - **args**: An object that describes optional parameters for the transfer
        * **delete**: A boolean corresponding to `rsync`'s `--delete` option. If true, `rsync` will remove remote files that no longer exist locally. Defaults to false.
        * **exclude**: A string or array of strings corresponding to `rsync`'s `--exclude` option. Defaults to None.

Example:

```
{
    "instruction": "SSH_GetFile",
    "wait": 0,
    "data": {
        "name": "workstation",
        "remote_path": "./data/output.dat",
        "local_path": "./results.txt"
    }
}
```

SSH_GetFTP

Downloads a path (file or directory) from the FTP server to the local filesystem.

- **data**: an object where the key is the argument name, and the value is the argument value
    - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
    - **host**: The hostname or address of the FTP server
    - **remote_path**: The path to the source file or directory on the FTP server

  - **local_path**: The local path to where the file or directory is to be downloaded

Example:

```
{
    "instruction": "SSH_GetFTP",
    "wait": 0,
    "data": {
        "name": "workstation",
        "host": "ftphost",
        "remote_path": "./data/output.dat",
        "local_path": "./results.txt"
    }
}
```

SSH_PutFTP

Uploads a path (file or directory) from the local filesystem to the FTP server.

- **data**: an object where the key is the argument name, and the value is the argument value
  - **name**: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
  - **host**: The hostname or address of the FTP server
  - **remote_path**: The path on the FTP server to where the file or directory is to be uploaded
  - **local_path**: The local path to the source file or directory

Example:

```
{
    "instruction": "SSH_PutFTP",
    "wait": 0,
    "data": {
        "name": "workstation",
        "host": "ftphost",
        "remote_path": "./data/output.dat",
        "local_path": "./results.txt"
    }
}
```

7 UserIO Plugin

The UserIO Plugin handles user input/output operations, including pausing test for safety critical operations. In such cases, CTF will wait until users confirm whether to continue or to abort the tests.

WaitForUserInput

When CTF executes WaitForUserInput, it will pause and wait for user input from stdin. If a user enters "Y", CTF will continue to execute next test instructions. Any other input will abort the test.

- **prompt**: optional value (example: "safety critical")

Example:

```
{
    "instruction":"WaitForUserInput",
    "data":{
     "prompt": "  "
    }
 }
```

8 Validation Plugin

The Validation Plugin provides functionality to interpret a cFE binary event log to a human-readable text file, and search for a text string in a file. It also allows the user to delete files/folders and copy files/folders on the local host machine.

SaveFileAsText

Save the cFE event log file (binary file created via the CFE_EVS_WRITE_LOG_DATA_FILE_CC command) to a human-readable text file

- **input_file**: path of the binary event log file.
- **output_file**: path of the output text file.
- **file_type**: currently only supports 'EVS' file type.
- **target**: cfs target, optional

Example:

```
{
    "instruction":"SaveFileAsText",
    "data":{
        "input_file": "/dev/shm/osal:RAM/event_log.bin",
        "output_file": "./testArtifacts/event_log.txt",
        "file_type": "EVS",
        "target": "cfs"
    }
 }
```

SearchStr

Search a text file for a given text string. If the string is found, return True, otherwise return False.

- **file**: path of the text file.
- **search_str**: text string to be searched for.

Example:

```
{
    "instruction":"SearchStr",
     "data":{
         "file": "/testArtifacts/event_log.txt",
         "search_str": "./testArtifacts/"
     }
}
```

CopyFiles

Copy a file or folder on the host file system. The source may point to a file or folder. If the destination exists, it will be overridden.

- **source**: path of the file / folder to be copied from on host machine.
- **destination**: path of the file / folder to be copied to on host machine.

Example:

```
{
    "instruction":"CopyFiles",
    "data":{
         "source": "./testArtifacts/",
         "destination": "./testArtifacts/"
     }
}
```

DeleteFiles

Delete a file or folder on the host file system.

- **path**: path of the file / folder to be deleted on host machine.

Example:

```
{
     "instruction": "DeleteFiles",
     "data": {
         "path": "./testArtifacts/"
     }
}
```

9 Variable Plugin

The Variable Plugin provides CTF users with the capability to Set / Check / Update user-defined variables from json test scripts. The defined variables can be used in control flow instructions, such as "BeginLoop" and "EndLoop". The following instructions are available.

SetUserVariable

Set / update the value of user defined variable.

- **variable_name**: the user-defined variable name (example: "my_var")
- **operator**: the operator applied to variable, including "=", "+", "-", "*", "/" (example: "=")
- **value**: numerical number (example: 0)

Example:

```
{
    "instruction": "SetUserVariable",
    "data": {
         "variable_name": "my_var",
          "operator": "=",
          "value": 0
    }
}
```

SetUserVariableFromTlm

Set the user defined variable to the latest telemetry value.

- **variable_name**: the user-defined variable name (example: "my_var")
- **mid**: the mid of telemetry packet (example: "TO_HK_TLM_MID")
- **tlm_variable**: the parameter of telemetry packet (example: "usCmdCnt")

Example:

```
{
    "instruction": "SetUserVariableFromTlm",
    "data": {
         "variable_name": "my_var",
          "mid": "TO_HK_TLM_MID",
          "tlm_variable": "usCmdCnt"
    }
}
```

SetUserVariableFromTlmHeader

Same as `SetUserVariableFromTlm` except the variable references the packet header.

- **variable_name**: the user-defined variable name (example: "my_var")
- **mid**: the mid of telemetry packet (example: "TO_HK_TLM_MID")
- **header_variable**: the parameter of telemetry packet (example: "pheader.length")

Example:

```
{
    "instruction": "SetUserVariableFromTlmHeader",
    "data": {
         "variable_name": "my_var",
          "mid": "TO_HK_TLM_MID",
          "tlm_variable": "pheader.length"
    }
}
```

CheckUserVariable

Compare the user-defined variable with the value using the operator. Return the bool outcome of the operation performed on the variables and values.

- **variable_name**: the user-defined variable name (example: "my_var")
- **operator**: the operator applied to variable, including "<", "<=",">", ">=", "==", "!=" (example: "==")

- **value**: numerical number (example: 4)

Example:

```
{
    "instruction": "CheckUserVariable",
    "data": {
         "variable_name": "my_var",
          "operator": "==",
          "value": 4
    }
}
```

10 Namespace Documentation

10.1 lib Namespace Reference

Namespaces

- args_validation
- ctf_global
- ctf_utility
- event_types
- exceptions
- ftp_interface
- logger
- plugin_manager
- readers
- script_manager
- status
- status_manager
- test
- test_script
- time_interface

10.1.1 Detailed Description

```
@namespace lib
CTF Core Components
```

10.2 lib.args_validation Namespace Reference

Data Structures

- class ArgsValidation

10.2.1 Detailed Description

```
@namespace lib.args_validation
Argument Validation Helper Utilities
```

10.3 lib.ctf_global Namespace Reference

Data Structures

- class CtfVerificationStage
- class Global

Variables

- string DEFAULT_CONFIG = "configs/default_config.ini"
  Default Config used by CTF if no config_file is provided in the arguments.

10.3.1 Detailed Description

```
@namespace lib.ctf_global
Exposes CTF global state information for utilization by CTF Plugins.

Global Test Info object accessible by all plugins.
Populated by script reader with test header
info and other useful values for plugins
```

10.4 lib.ctf_utility Namespace Reference

Functions

- def expand_path
- def switch_to_cft_directory
- def get_current_instruction_index
- def set_goto_instruction_index
- def set_variable
- def get_variable
- def resolve_variable
- def rgetattr

Variables

- dictionary **operator_map**
- string **MACRO_MARKER** = '#'
- string **VAR_MARKER** = '$'
- string **INDEX_PATTERN** = r'\[(.*?)\]'

10.4.1 Detailed Description

```
@namespace lib.ctf_utility
Utility library functions
```

10.4.2 Function Documentation

10.4.2.1 def lib.ctf_utility.expand_path (*path*)

```
Given a directory path, expand the path with the user directory and variables, returning the expanded path
```

10.4.2.2 def lib.ctf_utility.get_current_instruction_index ( )

```
Return the current instruction execution index
```

10.4.2.3 def lib.ctf_utility.get_variable (*variable_name*)

```
Get the user defined variable, which will be used in variable plugin
```

10.4.2.4 def lib.ctf_utility.resolve_variable (*variable*)

```
A variable may be passed to an instruction argument as a string "xyz$variable_name$abc",
Search the global variable_store to evaluate its value.
```

10.4.2.5 def lib.ctf_utility.rgetattr (*obj*, *attr*, *args*)

```
Given an object and an attribute name, return the value of the specified attribute.
```

10.4.2.6 def lib.ctf_utility.set_goto_instruction_index (*index*)

```
Set the instruction execution index in Global, which will be used in the ControlFlow Plugin
```

10.4.2.7 def lib.ctf_utility.set_variable (*variable_name*, *op_code*, *value*)

```
Set/Update the user defined variable, which will be used in ControlFlow and Variable Plugins
```

10.4.2.8 def lib.ctf_utility.switch_to_cft_directory ( )

```
Switch the working directory to ctf directory, return ctf directory path
```

10.4.3 Variable Documentation

10.4.3.1 dictionary lib.ctf_utility.operator_map

Initial value:

```
= {
    "+": operator.add,
    "-": operator.sub,
    "*": operator.mul,
    "/": operator.truediv,
    "<": operator.lt,
    "<=": operator.le,
    ">": operator.gt,
    ">=": operator.ge,
    "==": operator.eq,
    "!=": operator.ne,
}
```

10.5 lib.event_types Namespace Reference

Data Structures

- class Instruction

10.5.1 Detailed Description

```
@namespace lib.event_types
Event Type definitions for CTF
```

10.6 lib.exceptions Namespace Reference

Data Structures

- class CtfTestError
- class CtfConditionError
- class CtfParameterError

10.6.1 Detailed Description

```
@namespace lib.exceptions
Exception definitions for CTF
```

10.7 lib.ftp_interface Namespace Reference

Data Structures

- class FtpInterface

10.7.1 Detailed Description

```
@namespace lib.ftp_interface
FTP interface for CTF
```

10.8 lib.logger Namespace Reference

Data Structures

- class CtfLogLevel
- class TestFormatter

Functions

- def test
- def init_logger
- def set_logger_options_from_config
- def change_log_file

Variables

- **colorlog** = None
- string **LOG_FORMAT** = '[%(asctime)s.%(msecs)03d] %(module)-32s(%(lineno)-3d) *** %(levelname)s: %(message)s'
- string **TIME_FORMAT** = '%H:%M:%S'

- tuple **logger** = logging.getLogger()
- tuple **test_formatter** = TestFormatter(LOG_FORMAT)

10.8.1 Detailed Description

```
@namespace lib.logger
Logger configuration and initialization for CTF logging
```

10.8.2 Function Documentation

10.8.2.1 def lib.logger.change_log_file (*new_log_file*)

```
change_log_file function: Change log file to store logging information.
@param new_log_file: the new file for logger to store logging information.
@return None
```

10.8.2.2 def lib.logger.init_logger (*config*)

```
Initializes the logger with CTF-specific handlers and formatting
```

10.8.2.3 def lib.logger.set_logger_options_from_config (*config*)

```
Configures the logger, and sets the log directory and first log file for this test run
```

10.8.2.4 def lib.logger.test (*self*, *passed*, *cont*, *msg*, *args*, *kwargs*)

```
Passed as a callback to logging configuration for logging test results
@note - self is an instance of class Logger
```

10.9 lib.plugin_manager Namespace Reference

Data Structures

- class ArgTypes
- class Plugin
- class PluginManager

10.9.1 Detailed Description

```
@namespace lib.plugin_manager
The Plugin Manager is a CTF core component that manages CTF plugins.
```

10.10 lib.readers Namespace Reference

Namespaces

- json_script_reader

10.10.1 Detailed Description

```
@namespace lib.readers
CTF Json Script Reader
```

10.11 lib.readers.json_script_reader Namespace Reference

Data Structures

- class JSONScriptReader

10.11.1 Detailed Description

```
@namespace lib.readers.json_script_reader
Loads and validates input CTF test scripts. Manages execution of loaded test scripts.
```

10.12 lib.script_manager Namespace Reference

Data Structures

- class ScriptManagerConfig
- class ScriptManager

10.12.1 Detailed Description

```
@namespace lib.script_manager
Loads and manages test scripts during a test run
```

10.13 lib.status Namespace Reference

Data Structures

- class StatusDefs
- class ObjectFactory

10.13.1 Detailed Description

```
@namespace lib.status
Defines status messages to be sent out by CTF during a test run
```

10.14 lib.status_manager Namespace Reference

Data Structures

- class StatusManager

10.14.1 Detailed Description

```
@namespace lib.status_manager
Publishes CTF status messages over a UDP socket (utilized by the CTF editor)
```

10.15 lib.test Namespace Reference

Data Structures

- class Test

10.15.1 Detailed Description

```
@namespace lib.Test
Represents a single CTF test case
```

10.16 lib.test_script Namespace Reference

Data Structures

- class TestScript

10.16.1 Detailed Description

```
@namespace lib.test_script
Loads and validates input CTF test scripts. Manages execution of loaded test scripts.
```

10.17 lib.time_interface Namespace Reference

Data Structures

- class TimeInterface

10.17.1 Detailed Description

```
@namespace lib.time_interface
Interface definition for time managers to implement
```

10.18 plugins.cfs.cfs_config Namespace Reference

Data Structures

- class CfsConfig
- class RemoteCfsConfig

Variables

- **CONFIG** = Global.config

10.18.1 Detailed Description

```
cfs_config.py: CFS Plugin Config for CTF.

- Defines the expected fields in the cFS config section for
  a base (linux) target, as well as Remote SSH targets.
```

10.19 plugins.control_flow_plugin.control_flow_plugin Namespace Reference

Data Structures

- class ControlFlowPlugin

10.19.1 Detailed Description

```
@namespace plugins.control_flow_plugin
The Control-Flow Plugin provides the functionality of CTF control flow statement,
including looping and conditional statements.
```

10.20 plugins.example_plugin.example_plugin Namespace Reference

Data Structures

- class ExamplePlugin

10.20.1 Detailed Description

```
@namespace plugins.example_plugin
The Example Plugin module shows a minimal plugin implementation to be used as a
template for other CTF plugins
```

10.21 plugins.ssh.ssh_plugin Namespace Reference

Data Structures

- class SshConfig
- class SshPlugin
- class SshController

10.21.1 Detailed Description

```
@namespace plugins.ssh_plugin
The SSH Plugin provides remote and local shell command execution capability for CTF.
The module defines SshPlugin class and SshConfig, SshController helper class.
```

10.22 plugins.variable_plugin.variable_plugin Namespace Reference

Data Structures

- class VariablePlugin

10.22.1 Detailed Description

```
@namespace plugins.variable_plugin
The Variable Plugin module allows users to set / update / check variables defined in json test scripts.
```

11 Data Structure Documentation

11.1 lib.args_validation.ArgsValidation Class Reference

Public Member Functions

- def __init__
- def add_error
- def get_error_count
- def increment_error_count
- def verify_symbol
- def validate_symbol
- def validate_file
- def validate_number
- def validate_int
- def validate_ip
- def validate_boolean

Static Public Member Functions

- def is_param_none
- def validate_directory

Data Fields

- **parameter_errors**

11.1.1 Detailed Description

```
Helper class to validate arguments and data used by CTF
```

11.1.2 Constructor & Destructor Documentation

11.1.2.1 def lib.args_validation.ArgsValidation.__init__ (*self*)

```
Constructor of ArgsValidation class. Initialize instance variable "parameter_errors",
which tracks the number of errors encountered during validation.
```

11.1.3 Member Function Documentation

11.1.3.1 def lib.args_validation.ArgsValidation.add_error (*self, field, exception* = None)

```
Increment the number of errors and log an exception if needed
@param field: Field name where validation error occurred
@param exception:  Whether to log an exception on failure or not
```

11.1.3.2 def lib.args_validation.ArgsValidation.get_error_count (*self*)

```
Returns the number of errors encountered during validation so far
```

11.1.3.3 def lib.args_validation.ArgsValidation.increment_error_count (*self*)

```
Increment error count without logging an exception
```

11.1.3.4 def lib.args_validation.ArgsValidation.is_param_none (*param*) [static]

```
Returns whether or not a given parameter is None
@param param: Parameter to check if None
```

11.1.3.5 def lib.args_validation.ArgsValidation.validate_boolean (*self*, *value*)

```
Verify that the given value is valid as a boolean.
Return the converted value, or None if not a boolean.
```

11.1.3.6 def lib.args_validation.ArgsValidation.validate_directory (*directory*) [static]

```
Given a directory path, verify that the directory exists on disk.
Return the expanded absolute path, or None if invalid.
```

11.1.3.7 def lib.args_validation.ArgsValidation.validate_file (*self*, *file_path*, *fail_if_not_valid* = False)

```
Given a file path, verify that the file exists on disk.
Return the expanded absolute path, or None if invalid.

@param file_path: Path to file to check
@param fail_if_not_valid: Whether to consider an invalid path a failure or not
@note fail_if_not_valid is useful when checking a file that is not guaranteed to exist
```

11.1.3.8 def lib.args_validation.ArgsValidation.validate_int (*self*, *integer*)

```
Verify that a given value is valid as an integer.
Return the converted value, or None if not an integer.
```

11.1.3.9 def lib.args_validation.ArgsValidation.validate_ip (*self*, *ip_address*)

```
Verify that the given value is a valid and reachable network destination.
Return the IP address, or None if invalid.
```

11.1.3.10 def lib.args_validation.ArgsValidation.validate_number (*self*, *number*)

```
Verify that a given value is valid as a numerical (float).
Return the converted value, or None if not a number.
```

11.1.3.11 def lib.args_validation.ArgsValidation.validate_symbol (*self*, *symbol*, *file_path*)

```
Given a file path, verify that the file exists on disk and contains the given symbol.
Return the symbol if it exists, otherwise return None

@note - Primarily used to validate SP0 executables

@param symbol: Name of symbol to verify within executable file
@param file_path: Path to executable file to check
```

11.1.3.12 def lib.args_validation.ArgsValidation.verify_symbol (*self*, *file_path*, *symbol*)

```
Given a file path, verify that a given symbol exists within that file.
Return True if the symbol exists, otherwise return False

@note - Primarily used to validate SP0 executables

@param file_path: Path to executable file to check
@param symbol: Name of symbol to verify within executable file
```

11.2 lib.plugin_manager.ArgTypes Class Reference

Static Public Attributes

- string **cmd_mid** = "cmd_mid"
- string **cmd_code** = "cmd_code"
- string **cmd_arg** = "cmd_arg"
- string **tlm_mid** = "tlm_mid"
- string **comparison** = "comparison"
- string **string** = "string"
- string **boolean** = "boolean"
- string **number** = "number"
- string **ignore** = "ignore"
- string **condition** = "loop_condition"
- string **event** = "event"
- string **other** = "other"
- list **array_types** = [cmd_arg, comparison, condition, event]

11.2.1 Detailed Description

```
Argument types support by plugin instructions. The argument types are exported to the CTF Editor for autosuggesti
and input validation.
```

11.3 plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader Class Reference

Public Member Functions

- def **__init__**
- def process_command
- def process_telemetry
- def process_types
- def process_types_second_pass
- def process_custom_types
- def process_ccsds_json_file
- def get_ccsds_messages_from_dir

Static Public Member Functions

- def is_command_msg
- def is_telemetry_msg
- def is_command_tlm

- def is_types_macros
- def is_custom_types
- def validate_json_schema

Data Fields

- **current_file_name**
- **type_dict**
- **ctype_structure**

Private Member Functions

- def _build_data_type_and_field
- def _create_parameterized_type

11.3.1 Detailed Description

```
This class reads CCSDS export files in JSON format and creates dictionaries mapping names to Python types and val
```

11.3.2 Member Function Documentation

11.3.2.1 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader._build_data_type_and_field (*self*, *param*, *fields*, *subtypes* = None) [private]

```
Builds a field, containing a simple data type, for a custom type.
Returns the data type and appends it to fields.

@note - This method does not create or modify any types. The return value, and the in-out parameter fields,
should be used to create the type with create_type_class.

@param param: A dictionary containing JSON data defining a field of a parent type
@param fields: A list of fields of the parent type
@param subtypes: A dictionary of subtypes of the parent type
```

11.3.2.2 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader._create_parameterized_type (*self*, *json_dict*, *type_id* = None, *arg_id* = None, *subtypes* = None) [private]

```
Recursively creates custom type definitions from JSON data and any known subtypes,
and adds them to the type dictionary. Returns the top-level type and a dictionary of any enumerations.

@param json_dict: A dictionary containing JSON data defining a data type
@param type_id: The dictionary key for the name of the type
@param arg_id: The dictionary key for the definitions of subtypes, if any
@param subtypes: A dictionary mapping names of subtypes to their types, used in recursive calls
```

11.3.2.3 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.get_ccsds_messages_from_dir (*self*, *directory*)

```
Walks through a directory and parses CCSDS command and telemetry messages and type macros
from the JSON, as appropriate. Creates and returns dictionaries mapping names to these constructs.

@param directory: The path to the root directory containing CCSDS exports as .json files
```

11.3.2.4 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.is_command_msg (*json_data*) [static]

```
Returns whether a JSON dictionary represents a CCSDS command message.
```

11.3.2.5 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.is_command_tlm (*json_data*) [static]

```
Returns whether a JSON dictionary represents a CCSDS command or telemetry message.
```

11.3.2.6 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.is_custom_types (*json_data*) [static]

```
Returns whether a JSON dictionary represents custom type definitions.
```

11.3.2.7 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.is_telemetry_msg (*json_data*) [static]

```
Returns whether a JSON dictionary represents a CCSDS telemetry message.
```

11.3.2.8 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.is_types_macros (*json_data*) [static]

```
Returns whether a JSON dictionary represents type aliases or macros.

@note - A list is assumed to be type aliases or macros.
```

11.3.2.9 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.process_ccsds_json_file (*self*, *filename*, *file_filter* = None, *second_pass* = False)

```
Reads JSON from a single file and, if it matches the filter, parses the contents

@note - Because of interdependency between files, it is necessary to parse macros first for literal values,
then command and telemetry message types, then macros again for type aliases.

@param filename: The path to the file to be read
@param file_filter: A callable that will return True if the file is to be parsed. Pass None to parse all files
@param second_pass: True if this is the second time parsing type macros, whose types should already be defined
```

11.3.2.10 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.process_command (*self*, *json_dict*)

```
Parses the contents of a JSON dictionary for a CCSDS command message to dynamically create a new type for each
command code which is added to the type dictionary. Defines a command message with the MID and command codes,
and an enumeration for each command code by name.

@param json_dict: A dictionary containing the JSON data of an exported CCSDS telemetry message
```

11.3.2.11 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.process_custom_types (*self*, *json_dict*)

```
Parses the contents of a JSON dictionary for a custom data type which is added to the type dictionary. This type
may then be referenced by name in other files without redefining its structure.

@note - json_dict must include the keys "data_type" for the name and "parameters" for the contents regardless
of whether the data type is to be used in commands or telemetry.

@note - This method should be called before processing any command and telemetry messages so that the
definitions of these custom types are known.

@param json_dict: A dictionary containing the JSON data of an exported data type definition
```

11.3.2.12 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.process_telemetry (*self*, *json_dict*)

```
Parses the contents of a JSON dictionary for a CCSDS telemetry message to dynamically create a new type
which is added to the type dictionary. Defines a telemetry message with the MID, name, and type.

@param json_dict: A dictionary containing the JSON data of an exported CCSDS telemetry message
```

11.3.2.13 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.process_types (*self*, *json_list*)

```
Parses the contents of a JSON dictionary for type macros, and inserts any aliases, constants, or MID
mapping into the appropriate dictionaries.

@note - Only aliases of ctypes which are in the type dictionary will be processed. To process aliases of
custom types defined in other files, use process_types_second_pass after processing those files.

@param json_list: A dictionary containing the JSON data of exported CCSDS types
```

11.3.2.14 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.process_types_second_pass (*self*, *json_list*)

```
Parses the contents of a JSON dictionary for type aliases only, and adds them to the type dictionary if they
are not already defined.

@note - This method should be called after CCSDS command and telemetry messages have been processed so that
the definitions of those custom types are known.

@param json_list: A dictionary containing the JSON data of exported CCSDS types
```

11.3.2.15 def plugins.ccsds_plugin.readers.ccdd_export_reader.CCDDExportReader.validate_json_schema (*json_data*, *schema_path*) [static]

```
Validates a dictionary of JSON data against a schema file.

@param json_data: A dictionary containing JSON data to be validated
@param schema_path: Path to a JSON schema file
```

11.4 plugins.ccsds_plugin.ccsds_interface.CCSDSInterface Class Reference

Public Member Functions

- def **__init__**
- def add_telem_msg
- def add_cmd_msg
- def add_enumeration
- def get_ccsds_messages_from_dir

Data Fields

- **mids**
- **mid_map**
- **enum_map**
- **config**
- **header_info_included**
- **log_ccsds_imports**

11.4.1 Detailed Description

```
This class provides an interface and partial implementation for a CCSDS reader to process CCSDS data from
a directory into dynamic type definitions. The method of parsing the data is left to a subclass.
```

11.4.2 Member Function Documentation

11.4.2.1 def plugins.ccsds_plugin.ccsds_interface.CCSDSInterface.add_cmd_msg (*self*, *mid_name*, *mid*, *command_code_map*, *command_enums* = `None`)

```
Adds a command message to the internal types

@param mid_name: Name of the MID associated with the command
@param mid: Value of the MID associated with the command
@param command_code_map: Dictionary mapping command code values to their corresponding types
@param command_enums: Dictionary of enumerations associated with this command
```

11.4.2.2 def plugins.ccsds_plugin.ccsds_interface.CCSDSInterface.add_enumeration (*self*, *key*, *value*)

```
Adds an enumeration definition to the internal types

@param key: Name of the enumeration
@param value: Value of the enumeration
```

11.4.2.3 def plugins.ccsds_plugin.ccsds_interface.CCSDSInterface.add_telem_msg (*self*, *mid_name*, *mid*, *name*, *parameters*, *parameter_enums* = `None`)

```
Adds a telemetry message to the internal types

@param mid_name: Name of the MID associated with the command
@param mid: Value of the MID associated with the command
@param name: Name of the telemetry message
@param parameters: Type of the telemetry message parameters
@param parameter_enums: Dictionary of enumerations associated with this telemetry message
```

11.4.2.4 def plugins.ccsds_plugin.ccsds_interface.CCSDSInterface.get_ccsds_messages_from_dir (*self*, *directory*)

```
Virtual function to be implemented by a reader.
Processes the CCSDS data from a directory and returns the data types defined in them.

@param directory: Path to the directory containing CCSDS data type definitions.
```

11.5 plugins.ccsds_plugin.ccsds_packet_interface.CcsdsPacketInterface Class Reference

Public Member Functions

- def get_msg_id
- def **set_msg_id**
- def has_secondary_header
- def get_function_code
- def **set_function_code**

11.5.1 Detailed Description

```
This class provides a common interface for CCSDS packets to get and set values in the headers
```

```
without knowing where they are defined

@note - Classes implementing interface for specific CCSDS packets should inherit from this type
and override all methods.
```

11.5.2 Member Function Documentation

11.5.2.1 def plugins.ccsds_plugin.ccsds_packet_interface.CcsdsPacketInterface.get_function_code (*self*, *int*)

```
Convenience method to get the function code from the packet
```

11.5.2.2 def plugins.ccsds_plugin.ccsds_packet_interface.CcsdsPacketInterface.get_msg_id (*self*, *int*)

```
Convenience method to get the message ID from the packet
```

11.5.2.3 def plugins.ccsds_plugin.ccsds_packet_interface.CcsdsPacketInterface.has_secondary_header (*self*, *bool*)

```
Convenience method to check for the presence of a secondary header
```

11.6 plugins.ccsds_plugin.ccsds_packet_interface.CcsdsPacketType Class Reference

Static Public Attributes

- int **CommandPacket** = 1
- int **TelemetryPacket** = 0

11.6.1 Detailed Description

```
This class enumerates CCSDS packet types as integer values
```

11.7 plugins.ccsds_plugin.ccsds_plugin.CCSDSPlugin Class Reference

Public Member Functions

- def **__init__**
- def get_cfs_plugin
- def **initialize**
- def validate_cfs_ccsds_data
- def **shutdown**

Data Fields

- **name**
- **description**
- **command_map**
- **cfs_plugin**

Private Member Functions

- def **_send_commands_to_target**

11.7.1 Detailed Description

```
The CCSDS Plugin provides CCSDS validation support for CTF
```

11.7.2 Member Function Documentation

11.7.2.1 def plugins.ccsds_plugin.ccsds_plugin.CCSDSPlugin.get_cfs_plugin (*self*)

```
Returns the instance of the CFS Plugin registered with the plugin manager
```

11.7.2.2 def plugins.ccsds_plugin.ccsds_plugin.CCSDSPlugin.validate_cfs_ccsds_data (*self*, *target* = `None`)

```
Validates the CCSDS data types by sending an empty instance of each command code found in the MID map to CFS.

@note - This instruction will cause commands to be sent to the designated CFS target

@note - The plugin cannot directly verify that CFS is able to process the received data.
CFS output should be checked to ensure that no invalid length commands were received.

@param target: The name of the CFS target to be used for validation.
     If not provided, the default target will be used.
```

11.8 plugins.ccsds_plugin.ccsds_primary_header.CcsdsPrimaryHeaderBase Class Reference

Public Member Functions

- def __init__
- def **set_ccsds_version**
- def **set_app_id**
- def **set_secondary_header_flag**
- def **set_segmentation_flags**
- def **set_sequence_count**
- def **set_packet_length**
- def **set_packet_type**
- def is_command
- def get_msg_id

Data Fields

- **version_number**
- **app_id**
- **secondary_header_flag**
- **segmentation_flags**
- **sequence_count**
- **length**
- **type**

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.8.1 Detailed Description

```
This class implements the CCSDS primary header as represented by a ctypes BigEndianStructure
```

11.8.2 Constructor & Destructor Documentation

11.8.2.1 def plugins.ccsds_plugin.ccsds_primary_header.CcsdsPrimaryHeaderBase.__init__ (*self*)

```
class CcsdsPrimaryHeaderBase constructor: assign attributes default values
```

11.8.3 Member Function Documentation

11.8.3.1 def plugins.ccsds_plugin.ccsds_primary_header.CcsdsPrimaryHeaderBase.get_msg_id (*self, int*)

```
Returns the message ID value derived from the header fields
```

11.8.3.2 def plugins.ccsds_plugin.ccsds_primary_header.CcsdsPrimaryHeaderBase.is_command (*self, int*)

```
Returns true if the packet represents a command, indicated by the type field
```

11.8.4 Field Documentation

11.8.4.1 list plugins.ccsds_plugin.ccsds_primary_header.CcsdsPrimaryHeaderBase._fields_ [static], [private]

Initial value:

```
= [
        ("version_number", ctypes.c_uint16, 3),  # CCSDS version
        ("type", ctypes.c_uint16, 1),  # Packet type:      0 = TLM, 1 = CMD
        ("secondary_header_flag", ctypes.c_uint16, 1),  # Secondary header: 0 = absent, 1 = present
        ("app_id", ctypes.c_uint16, 11),  # Application ID
        ("segmentation_flags", ctypes.c_uint16, 2),  # Segmentation flags:  3 = complete packet
        ("sequence_count", ctypes.c_uint16, 14),  # Sequence count
        ("length", ctypes.c_uint16, 16)  # (total packet length) - 7
    ]
```

11.9 plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryCmdHeader Class Reference

Public Member Functions

- def __init__
- def **set_function_code**
- def **set_checksum**
- def get_function_code
- def get_checksum

Data Fields

- **checksum**
- **function_code**

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.9.1 Detailed Description

```
This class implements the CCSDS secondary header as represented by a ctypes BigEndianStructure
```

11.9.2 Constructor & Destructor Documentation

11.9.2.1 def plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryCmdHeader.__init__ (*self*)

```
class CcsdsSecondaryCmdHeader constructor: assign attributes default values
```

11.9.3 Member Function Documentation

11.9.3.1 def plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryCmdHeader.get_checksum (*self, int*)

```
Gets the checksum value
```

11.9.3.2 def plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryCmdHeader.get_function_code (*self, int*)

```
Gets the function code value
```

11.9.4 Field Documentation

11.9.4.1 list plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryCmdHeader._fields_ [static], [private]

Initial value:

```
= [
        ("function_code", ctypes.c_uint8),
        ("checksum", ctypes.c_uint8)
    ]
```

11.10 plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryTlmHeader Class Reference

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.10.1 Detailed Description

```
This class implements the CCSDS secondary telemetry header as represented by a ctypes BigEndianStructure
```

11.10.2 Field Documentation

11.10.2.1 list plugins.ccsds_plugin.cfe.ccsds_secondary_header.CcsdsSecondaryTlmHeader._fields_ [static], [private]

Initial value:

```
= [
        ("timestamp_seconds", ctypes.c_uint32),
        ("timestamp_subseconds", ctypes.c_uint16),
    ]
```

11.11 plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1CmdPacket Class Reference

Public Member Functions

- def **__init__**
- def get_function_code
- def **set_function_code**
- def **set_checksum**

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.11.1 Detailed Description

```
This class implements a CCSDS V1 command packet as represented by a ctypes BigEndianStructure
```

11.11.2 Member Function Documentation

11.11.2.1 def plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1CmdPacket.get_function_code (*self*, *int*)

```
Convenience method to get the function code from the packet
```

11.11.3 Field Documentation

11.11.3.1 list plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1CmdPacket._fields_ [static], [private]

Initial value:

```
= [
        ("sheader", CcsdsSecondaryCmdHeader)
    ]
```

11.12 plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1Packet Class Reference

Public Member Functions

- def **set_msg_id**
- def get_msg_id
- def has_secondary_header

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.12.1 Detailed Description

```
This class provides an interface to a CCSDS V1 packet
```

11.12.2 Member Function Documentation

11.12.2.1 def plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1Packet.get_msg_id (*self, int*)

```
Convenience method to get the message ID from the packet
```

11.12.2.2 def plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1Packet.has_secondary_header (*self, bool*)

```
Convenience method to check for the presence of a secondary header
```

11.12.3 Field Documentation

11.12.3.1 list plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1Packet._fields_ [static], [private]

Initial value:

```
= [
        ("pheader", CcsdsV1PrimaryHeader)
    ]
```

11.13 plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1PrimaryHeader Class Reference

Additional Inherited Members

11.13.1 Detailed Description

```
This is a marker interface to indicate a CCSDS V1 primary header
```

11.14 plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1TlmPacket Class Reference

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

Additional Inherited Members

11.14.1 Detailed Description

```
This class implements a CCSDS V1 telemetry packet as represented by a ctypes BigEndianStructure
```

11.14.2 Field Documentation

11.14.2.1 list plugins.ccsds_plugin.cfe.ccsds_v1.ccsds_v1.CcsdsV1TlmPacket._fields_ [static], [private]

Initial value:

```
= [
        ("sheader", CcsdsSecondaryTlmHeader)
    ]
```

11.15 plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2CmdPacket Class Reference

Public Member Functions

- def **__init__**
- def **get_function_code**
- def **set_function_code**

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.15.1 Detailed Description

```
This class implements a CCSDS V2 command packet as represented by a ctypes BigEndianStructure
```

11.15.2 Field Documentation

11.15.2.1 list plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2CmdPacket._fields_ [static], [private]

Initial value:

```
= [
        ("sheader", CcsdsSecondaryCmdHeader)
    ]
```

11.16 plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2ExtendedHeader Class Reference

Public Member Functions

- def __init__
- def **set_eds_version**
- def **set_endian**
- def **set_playback_flag**
- def **set_subsystem_id**
- def **set_system_id**

Data Fields

- **eds_version**
- **endian**
- **playback_flag**
- **subsystem_id**
- **system_id**

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.16.1 Detailed Description

```
This class implements a CCSDS V2 extended header as represented by a ctypes BigEndianStructure
```

11.16.2 Constructor & Destructor Documentation

11.16.2.1 def plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2ExtendedHeader.__init__ (*self*)

```
class CcsdsV2ExtendedHeader constructor: assign attributes default values
```

11.16.3 Field Documentation

11.16.3.1 list plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2ExtendedHeader._fields_ [static], [private]

Initial value:

```
= [
        ("eds_version", ctypes.c_uint16, 5),  # EDS Version for packet definition used
        ("endian", ctypes.c_uint16, 1),  # Endian: Big = 0, Little = 1
        ("playback_flag", ctypes.c_uint16, 1),  # Playback flag 0 = original, 1 = playback
        ("subsystem_id", ctypes.c_uint16, 9),  # Subsystem ID (mission defined)
        ("system_id", ctypes.c_uint16, 16),  # System ID (mission defined)
    ]
```

11.17 plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2Packet Class Reference

Public Member Functions

- def **set_msg_id**
- def get_msg_id
- def **has_secondary_header**
- def **get_function_code**
- def **set_function_code**

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

11.17.1 Detailed Description

```
This class provides an interface to a CCSDS V2 packet
```

11.17.2 Member Function Documentation

11.17.2.1 def plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2Packet.get_msg_id (*self*, *int*)

```
Returns the message ID derived from the header fields
Python implementation of CFE_SB_GetMsgId(CFE_SB_MsgPtr_t MsgPtr)
```

11.17.3 Field Documentation

11.17.3.1 list plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2Packet._fields_ [static], [private]

Initial value:

```
= [
        ("pheader", CcsdsV2PrimaryHeader),
        ("eheader", CcsdsV2ExtendedHeader)
    ]
```

11.18 plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2PrimaryHeader Class Reference

Public Member Functions

- def **is_command**

Additional Inherited Members

11.18.1 Detailed Description

```
This class provides an interface to a CCSDS V2 primary header

@note - This is a sample implementation showing how custom headers can extend CcsdsPrimaryHeaderBase as needed.
        The implementation of is_command is redundant with CcsdsPrimaryHeaderBase.
```

11.19 plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2TlmPacket Class Reference

Static Private Attributes

- int **_pack_** = 1
- list **_fields_**

Additional Inherited Members

11.19.1 Detailed Description

```
This class implements a CCSDS V2 telemetry packet as represented by a ctypes BigEndianStructure
```

11.19.2 Field Documentation

11.19.2.1 list plugins.ccsds_plugin.cfe.ccsds_v2.ccsds_v2.CcsdsV2TlmPacket._fields_ [static], [private]

Initial value:

```
= [
        ("sheader", CcsdsSecondaryTlmHeader)
    ]
```

11.20 plugins.ccsds_plugin.ccsds_packet_interface.CcsdsVer Class Reference

Static Public Attributes

- int **Ccsds_ver_1** = 1
- int **Ccsds_ver_2** = 2
- int **Ccsds_ver_GW** = 3

11.20.1 Detailed Description

```
This class enumerates CCSDS versions as integer values
```

11.21 plugins.cfs.cfs_config.CfsConfig Class Reference

Public Member Functions

- def __init__
- def configure
- def load_field
- def load_config_data
- def set_ctf_ip
- def set_cfs_run_cmd
- def get_error_count

Data Fields

- **sections**
- **validation**
- **name**
- **cfs_protocol**
- **build_cfs**
- **ccsds_data_dir**
- **ccsds_target**
- **log_ccsds_imports**
- **cfs_build_dir**
- **cfs_build_cmd**
- **cfs_run_dir**
- **cfs_port_arg**
- **cfs_exe**
- **cfs_run_args**

- **cfs_ram_drive_path**
- **cfs_run_cmd**
- **cfs_output_file**
- **remove_continuous_on_fail**
- **cfs_target_ip**
- **ctf_ip**
- **cmd_udp_port**
- **tlm_udp_port**
- **evs_log_file**
- **cfs_debug**
- **cfs_run_in_xterm**
- **tlm_app_choice**
- **ccsds_ver**
- **evs_long_event_mid_name**
- **evs_short_event_mid_name**
- **evs_messages_clear_after_time**
- **endianess_of_target**
- **ccsds_header_info_included**
- **telemetry_debug**

11.21.1 Detailed Description

```
 The CFS Configuration classes handle interpreting the respective CFS Target section
 in the loaded INI config. The INI config could have multiple CFS targets, defined as sections,
 and each target specifies the needed fields.
 Documentation of the CFS configuration fields can be found in the CFS Plugin README. Refer to INI or README for
 field descriptions.
```

11.21.2 Constructor & Destructor Documentation

11.21.2.1 def plugins.cfs.cfs_config.CfsConfig.__init__ (*self*, *name*)

```
 Constructor for CfsConfig class.  Assign all Cfs config attributes to default None.
```

11.21.3 Member Function Documentation

11.21.3.1 def plugins.cfs.cfs_config.CfsConfig.configure (*self*, *name*)

```
Setup CfsConfig attributes based on INI file
```

11.21.3.2 def plugins.cfs.cfs_config.CfsConfig.get_error_count (*self*)

```
Return field validation error counts.
@return field validation error counts
```

11.21.3.3 def plugins.cfs.cfs_config.CfsConfig.load_config_data (*self*, *section_name*)

```
From loaded sections of INI config, interpret CFS target config attributes, including
build_cfs, CCSDS_data_dir, CCSDS_target, etc.
@param section_name: loaded Json CFS target section.
@return None
```

11.21.3.4 def plugins.cfs.cfs_config.CfsConfig.load_field (*self*, *section*, *field_name*, *config_getter*, *validate_function* = `None`)

```
Interpret field attribute of loaded CFS target config section.
@param section: loaded Json CFS target section.
@param field_name: the field name for loaded attribute.
@param config_getter: the function to get an option value for a given section.
@param validate_function: the function to validate the field attribute (Optional).
@return Any: field attribute with matching name
```

11.21.3.5 def plugins.cfs.cfs_config.CfsConfig.set_cfs_run_cmd (*self*, *cfs_exe* = "", *cfs_run_args* = "")

```
Set CFS config attribute cfs_exe, cfs_run_args, and cfs_run_cmd.
if passed arguments are empty string, use the config attributes from INI file
@param cfs_exe: CFS executable name. if it is empty string, use the field from INI file
@param cfs_run_args: CFS executable arguments. if it is empty string, use the field from INI file
@return None
```

11.21.3.6 def plugins.cfs.cfs_config.CfsConfig.set_ctf_ip (*self*)

```
Get the IP address through a temporary created socket to CFS target, and assign the ip to config attribute
@return None
```

11.22 plugins.cfs.pycfs.cfs_controllers.CfsController Class Reference

Public Member Functions

- def __init__
- def process_ccsds_files
- def initialize
- def build_cfs
- def start_cfs
- def enable_cfs_output
- def **send_cfs_command**
- def **build_command_payload**
- def convert_args_to_ctypes
- def **encode_ctypes_to_bytes**
- def resolve_macros
- def resolve_simple_type
- def resolve_args_from_dict
- def check_tlm_value
- def **get_tlm_value**
- def check_tlm_continuous
- def convert_check_tlm_args
- def remove_check_tlm_continuous
- def check_event
- def archive_cfs_files
- def shutdown_cfs
- def shutdown
- def validate_mid_value
- def validate_cc_value

Static Public Member Functions

- def field_class_by_name

Data Fields

- **config**
- **cfs_process_list**
- **cfs**
- **ccsds_reader**
- **mid_map**
- **macro_map**
- **ccsds**
- **first_call_flag**
- **mid_pkt_count**
- **cfs_running**

11.22.1 Detailed Description

```
CfsController class Definition: CFS Controller Implementation for CTF.

@note When the CFS plugin registers a target, a cFS controller object is instantiated.
@note After the cfs_plugin receives a test instruction, the cFS controller handles all
  lower-level functionality beneath the plugin.
@note On controller initialization, telem/command interfaces are established, CCSDS message
  definitions are parsed to build the mid map, and controller becomes ready to send commands
  and verify telemetry.
@note Controller implements the specific functionality needed to execute the cFS plugin instructions
@note Controller manages cFS process, and will shutdown the target at the end of the test script or on
     ShutdownCfs instruction.
```

11.22.2 Constructor & Destructor Documentation

11.22.2.1 def plugins.cfs.pycfs.cfs_controllers.CfsController.__init__ (*self*, *config*)

```
Constructor implementation for CfsController class. Assign default values for CfsController properties
```

11.22.3 Member Function Documentation

11.22.3.1 def plugins.cfs.pycfs.cfs_controllers.CfsController.archive_cfs_files (*self*, *source_path*)

```
Implementation of CFS plugin instructions archive_cfs_files. When CFS plugin instructions
(archive_cfs_files) is executed, it calls CfsController instance's archive_cfs_files function.
```

11.22.3.2 def plugins.cfs.pycfs.cfs_controllers.CfsController.build_cfs (*self*)

```
Implementation of CFS plugin instructions build_cfs. When CFS plugin instructions (build_cfs) is executed,
it calls CfsController instance's build_cfs function.
```

11.22.3.3 def plugins.cfs.pycfs.cfs_controllers.CfsController.check_event (*self*, *app_name*, *event_id*, *event_str* = None, *is_regex* = False, *event_str_args* = None)

```
Checks for an EVS event message in the telemetry packet history,
assuming a particular structure for CFE_EVS_LongEventTlm_t.
This can be generified in the future to determine the structure from the MID map.
```

11.22.3.4 def plugins.cfs.pycfs.cfs_controllers.CfsController.check_tlm_continuous (*self*, *v_id*, *mid*, *args*)

```
Implementation of CFS plugin instructions check_tlm_continuous. When CFS plugin instructions
(check_tlm_continuous) is executed, it calls CfsController instance’s check_tlm_continuous function.
```

11.22.3.5 def plugins.cfs.pycfs.cfs_controllers.CfsController.check_tlm_value (*self*, *mid*, *args* = None)

```
Implementation of CFS plugin instructions check_tlm_value. When CFS plugin instructions (check_tlm_value)
is executed, it calls CfsController instance’s check_tlm_value function.
```

11.22.3.6 def plugins.cfs.pycfs.cfs_controllers.CfsController.convert_args_to_ctypes (*self*, *args*, *arg_class*, *ctypes*, *Structure*)

```
Implements the conversion of command args into a ctypes structure
```

11.22.3.7 def plugins.cfs.pycfs.cfs_controllers.CfsController.convert_check_tlm_args (*self*, *args*)

```
Implementation of helper function convert_check_tlm_args.
Convert telemetry data args with "value" to a list
```

11.22.3.8 def plugins.cfs.pycfs.cfs_controllers.CfsController.enable_cfs_output (*self*)

```
Implementation of CFS plugin instructions enable_cfs_output.  When CFS plugin instructions
(enable_cfs_output) is executed, it calls CfsController instance’s enable_cfs_output function.
```

11.22.3.9 def plugins.cfs.pycfs.cfs_controllers.CfsController.field_class_by_name (*name*, *args_class*) [static]

```
Implementation of helper function field_class_by_name.
Return a field with matching name.
```

11.22.3.10 def plugins.cfs.pycfs.cfs_controllers.CfsController.initialize (*self*)

```
Initialize CfsController instance, including the followings: create mid map; import ccsds header;
create command interface; create telemetry interface; create local CFS interface
```

11.22.3.11 def plugins.cfs.pycfs.cfs_controllers.CfsController.process_ccsds_files (*self*)

```
Create mid map for CFS plugin, if map does not exist, create ccsds_reader from INIT config file.
```

11.22.3.12 def plugins.cfs.pycfs.cfs_controllers.CfsController.remove_check_tlm_continuous (*self*, *v_id*)

```
Implementation of CFS plugin instructions remove_check_tlm_continuous. When CFS plugin instructions
(remove_check_tlm_continuous) is executed, it calls CfsController instance’s function.
```

11.22.3.13 def plugins.cfs.pycfs.cfs_controllers.CfsController.resolve_args_from_dict (*self*, *args*, *args_class*)

```
Implementation of helper function resolve_args_from_dict.
Convert argument args to args_class
```

11.22.3.14 def plugins.cfs.pycfs.cfs_controllers.CfsController.resolve_macros (*self*, *arg*)

```
Implementation of helper function resolve_macros.
search macro_map to convert arg to string.
```

11.22.3.15 def plugins.cfs.pycfs.cfs_controllers.CfsController.resolve_simple_type (*self*, *arg*, *arg_type*)

```
Implementation of helper function resolve_simple_type.
Resolves any macros in arg and converts it to a type appropriate for arg_class
```

11.22.3.16 def plugins.cfs.pycfs.cfs_controllers.CfsController.shutdown (*self*)

```
This function will shut down the CFS application being tested even if the JSON test file does not
include the shutdown test command
```

11.22.3.17 def plugins.cfs.pycfs.cfs_controllers.CfsController.shutdown_cfs (*self*)

```
Implementation of CFS plugin instructions shutdown_cfs. When CFS plugin instructions
(shutdown_cfs) is executed, it calls CfsController instance’s shutdown_cfs function.
```

11.22.3.18 def plugins.cfs.pycfs.cfs_controllers.CfsController.start_cfs (*self*, *run_args*)

```
Implementation of CFS plugin instructions start_cfs. When CFS plugin instructions (start_cfs) is executed,
it calls CfsController instance’s start_cfs function.
```

11.22.3.19 def plugins.cfs.pycfs.cfs_controllers.CfsController.validate_cc_value (*self*, *mid_dict*, *cc*)

```
Implementation of helper function validate_cc_value.
Attempt to convert a value to a CC name and check that it is in the provided mid_dict
@return str: A valid CC name if found, else None
```

11.22.3.20 def plugins.cfs.pycfs.cfs_controllers.CfsController.validate_mid_value (*self*, *mid*)

```
Implementation of helper function validate_mid_value.
Attempt to convert a value to a MID name and check that it is in the mid_map
@return str: A valid MID name if found, else None
```

11.23 plugins.cfs.pycfs.cfs_interface.CfsInterface Class Reference

Public Member Functions

- def __init__
- def build_cfs
- def start_cfs
- def stop_cfs
- def write_tlm_log
- def write_evs_log
- def read_sb_packets
- def parse_command_packet
- def parse_telemetry_packet
- def log_unknown_packet_mid
- def log_invalid_packet
- def **on_packet_received**
- def add_tlm_condition
- def remove_tlm_condition
- def check_tlm_conditions
- def send_command
- def check_value

- def clear_received_msgs_before_verification_start
- def check_tlm_value
- def **get_tlm_value**
- def check_tlm_packet
- def enable_output

Static Public Member Functions

- def check_strings

Data Fields

- **config**
- **evs_long_event_msg_mid**
- **evs_short_event_msg_mid**
- **init_passed**
- **command**
- **telemetry**
- **mid_payload_map**
- **output_manager**
- **cfs_std_out_path**
- **evs_log_file**
- **tlm_log_file**
- **tlm_has_been_received**
- **unchecked_packet_mids**
- **tlm_verifications_by_mid_and_vid**
- **cmd_packet_list**
- **received_mid_packets_dic**
- **has_received_mid**
- **ccsds**
- **pheader_offset**
- **should_skip_header**
- **tlm_header_offset**
- **cmd_header_offset**

11.23.1 Detailed Description

```
CfsInterface: Base-class Lower-level interface to communicate with cFS.
```

11.23.2 Constructor & Destructor Documentation

11.23.2.1 def plugins.cfs.pycfs.cfs_interface.CfsInterface.__init__ (*self*, *config*, *telemetry*, *command*, *mid_map*, *ccsds*)

```
Constructor for CfsInterface class.  Assign config, telemetry, command, mid_map,
and ccsds arguments to interface attributes
```

11.23.3 Member Function Documentation

11.23.3.1 def plugins.cfs.pycfs.cfs_interface.CfsInterface.add_tlm_condition (*self*, *v_id*, *mid*, *args*)

```
Add verification condition (with ID) to telemetry verification dictionary and do verification based on id
```

11.23.3.2 def plugins.cfs.pycfs.cfs_interface.CfsInterface.build_cfs (*self*)

```
Abstract class method, raise NotImplementedError exception
```

11.23.3.3 def plugins.cfs.pycfs.cfs_interface.CfsInterface.check_strings (*actual, expected, equal*) [static]

```
Check whether string argument actual == string argument expected,
if yes, return argument equal, otherwise return not equal
```

11.23.3.4 def plugins.cfs.pycfs.cfs_interface.CfsInterface.check_tlm_conditions (*self*)

```
Check all unchecked telemetry message by mid and vid.
If verification fails, raise CtfConditionError exception.
```

11.23.3.5 def plugins.cfs.pycfs.cfs_interface.CfsInterface.check_tlm_packet (*self, payload, args*)

```
Check telemetry message's value based on argument payload and args
```

11.23.3.6 def plugins.cfs.pycfs.cfs_interface.CfsInterface.check_tlm_value (*self, mid, args =* None*, discard_old_packets =* True)

```
 Given a mid and a arguments, iterate over all received packets since the start of the verification.
 Validate each packet until a success is seen, or there are no more packets to check.
```

11.23.3.7 def plugins.cfs.pycfs.cfs_interface.CfsInterface.check_value (*self, actual, expected, compare, mask, mask_value*)

```
Based on the argument compare value, use different method to compare argument actual and expected
```

11.23.3.8 def plugins.cfs.pycfs.cfs_interface.CfsInterface.clear_received_msgs_before_verification_start (*self, mid*)

```
Given a mid argument, iterate over all received packets.
If packets' received time expires, clear the packets with matching mid.
```

11.23.3.9 def plugins.cfs.pycfs.cfs_interface.CfsInterface.enable_output (*self*)

```
Send a command to enable output and check if we receive a response
```

11.23.3.10 def plugins.cfs.pycfs.cfs_interface.CfsInterface.log_invalid_packet (*self, mid*)

```
If this is the first time receiving a packet with the given mid, log the packet.
```

11.23.3.11 def plugins.cfs.pycfs.cfs_interface.CfsInterface.log_unknown_packet_mid (*self, mid*)

```
If this is the first time receiving a packet with the given mid, log the message.
```

11.23.3.12 def plugins.cfs.pycfs.cfs_interface.CfsInterface.parse_command_packet (*self, buffer*)

```
Parse command packets from received buffer.
```

11.23.3.13 def plugins.cfs.pycfs.cfs_interface.CfsInterface.parse_telemetry_packet (*self, buffer*)

```
Parse telemetry packets from received buffer.
```

11.23.3.14 def plugins.cfs.pycfs.cfs_interface.CfsInterface.read_sb_packets (*self*)

```
read_sb_packets() is responsible for receiving packets coming from the CFS application that is being tested
and placing them in a dictionary of lists that is ordered by mids as shown below.
received_mid_packets_dic = {
    "mid1": ["The last packet received with mid1"],
    "mid2": ["The last packet received with mid2"]
 }
 }
```

11.23.3.15 def plugins.cfs.pycfs.cfs_interface.CfsInterface.remove_tlm_condition (*self*, *v_id*)

```
Remove verification condition (with ID) from telemetry verification dictionary.
```

11.23.3.16 def plugins.cfs.pycfs.cfs_interface.CfsInterface.send_command (*self*, *msg_id*, *function_code*, *data*, *header_args* = None)

```
Send instruction to CFS instance through command interface.
```

11.23.3.17 def plugins.cfs.pycfs.cfs_interface.CfsInterface.start_cfs (*self*, *run_args*)

```
Abstract class method, raise NotImplementedError exception
```

11.23.3.18 def plugins.cfs.pycfs.cfs_interface.CfsInterface.stop_cfs (*self*)

```
Stop CFS executable instance, close command and telemetry sockets.
```

11.23.3.19 def plugins.cfs.pycfs.cfs_interface.CfsInterface.write_evs_log (*self*, *payload*)

```
Write payload and mid to evs log file. if log file does not exist, create one.
```

11.23.3.20 def plugins.cfs.pycfs.cfs_interface.CfsInterface.write_tlm_log (*self*, *payload*, *buf*, *mid*)

```
Write payload and mid to telemetry log file. if log file does not exist, create one.
```

11.24 plugins.cfs.cfs_plugin.CfsPlugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def **register_cfs**
- def **load_configured_targets**
- def **get_cfs_targets**
- def **build_cfs**
- def **start_cfs**
- def **enable_cfs_output**
- def **send_cfs_command**
- def **send_raw_cfs_command**
- def **check_tlm_value**
- def **check_tlm_packet**
- def **check_no_tlm_packet**

- def **get_tlm_value**
- def **check_tlm_continuous**
- def **remove_check_tlm_continuous**
- def **check_event**
- def **check_noevent**
- def **shutdown_cfs**
- def **archive_cfs_files**
- def shutdown

Data Fields

- **name**
- **description**
- **targets**
- **has_attempted_register**
- **protocols**
- **command_map**
- **verify_required_commands**
- **continuous_commands**
- **end_test_on_fail_commands**

Static Public Attributes

- string **FALLBACK_TARGET_NAME** = 'cfs'

11.24.1 Detailed Description

```
The CFS Plugin provides CFS command/telemetry support for CTF.

 @note The CFS plugin draws many default values from the CTF config file.
        The section [cfs] defines defaults for all CFS targets and is always required.

 @note If multiple CFS targets are to be registered, for each target name,
        the plugin will load values from a correspondingly named section.

 @note If no targets are explicitly registered by name by the time StartCfs is first executed,
       the plugin will automatically configure targets for each config section beginning with cfs_.
       If no such sections are found, the plugin will configure a single target using the [cfs] config section.
       If the cfs_protocol field is not found in the cfs section, a local target will be registered.

 @note The precedence of values is first the named config section, if any, and then the [cfs] config section.
       A target cannot be registered, explicitly nor automatically, without a correspondingly named config sectio
```

11.24.2 Constructor & Destructor Documentation

11.24.2.1 def plugins.cfs.cfs_plugin.CfsPlugin.__init__ (*self*)

```
Constructor for CfsPlugin.
Most importantly populates the command map and verify required commands,
which serve as the interface to the plugin manager.
```

11.24.3 Member Function Documentation

11.24.3.1 def plugins.cfs.cfs_plugin.CfsPlugin.initialize (*self*, *bool*)

```
Initializes the plugin by creating the CfsTimeManager.
This method is intended to be called by the plugin manager before the test script runs.
```

11.24.3.2 def plugins.cfs.cfs_plugin.CfsPlugin.shutdown (*self*, *None*)

```
Shuts down the plugin, releasing target resources.
Only runs when the plugin itself is shutting down.
To shut down individual targets, use shutdown_cfs.
```

11.25 plugins.cfs.cfs_time_manager.CfsTimeManager Class Reference

Public Member Functions

- def __init__
- def wait
- def pre_command
- def run_continuous_verifications

Static Public Member Functions

- def handle_test_exception_during_wait

Data Fields

- **ctf_verification_poll_period**
- **cfs_targets**

11.25.1 Detailed Description

```
CfsTimeManager: CFS Time Manager for CTF.

@note When initialized by the cFS plugin, the default CTF time manager (OS Time)
is disabled, and the cFS time manager is used instead.

@note The cFS time manager implements a serialized telemetry receive implementation
as CTF instructions are "waiting".

@note The cFS time manager also invokes the continuous verification checks
between polls to ensure each packet is verified if a continuous verification exists.
```

11.25.2 Constructor & Destructor Documentation

11.25.2.1 def plugins.cfs.cfs_time_manager.CfsTimeManager.__init__ (*self*, *cfs_targets*)

```
Constructor implementation for CfsTimeManager class.
@note CfsTimeManager is inherited from TimeInterface class.
@note The constructor assigns ctf_verification_poll_period attribute based on INI File config, and cfs_targets
attribute from passed argument.
```

11.25.3 Member Function Documentation

11.25.3.1 def plugins.cfs.cfs_time_manager.CfsTimeManager.handle_test_exception_during_wait (*error*, *msg*, *do_raise* = False) [static]

```
Test exception handler, log error, and raise exception if do_raise is True
```

11.25.3.2 def plugins.cfs.cfs_time_manager.CfsTimeManager.pre_command (*self*)

```
Read Telemetry Packets for CFS Target, and run continuous verification.
Raise any occurring Exception
```

11.25.3.3 def plugins.cfs.cfs_time_manager.CfsTimeManager.run_continuous_verifications (*self*)

```
Check all unchecked telemetry message by mid and vid by triggering target's target.cfs.check_tlm_conditions().
Raise any occurring Exception
```

11.25.3.4 def plugins.cfs.cfs_time_manager.CfsTimeManager.wait (*self*, *seconds*)

```
Do polling for certain seconds. Continue to do pre_command(), post_command(),
and sleep until exec_time expires.

@param seconds: polling duration.

@return None
```

11.26 plugins.ccsds_plugin.readers.command_builder.CommandArg Class Reference

Public Member Functions

- def **__init__**
- def **__getattr__**
- def **__setattr__**
- def **__delattr__**

Data Fields

- **name**
- **data_type**

11.26.1 Detailed Description

```
Class representing a CCSDS Command Argument

@param name: argument name
@param data_type: argument type
```

11.27 plugins.ccsds_plugin.readers.command_builder.CommandCode Class Reference

Public Member Functions

- def **__init__**

- def **__getattr__**
- def **__setattr__**
- def **__delattr__**

Data Fields

- **cc_name**
- **cc_value**
- **args**

11.27.1 Detailed Description

```
Class representing a Command Code for a CCSDS Command

@param name: command code name
@param code: command code value
```

11.28 plugins.cfs.pycfs.command_interface.CommandInterface Class Reference

Public Member Functions

- def __init__
- def init_socket
- def cleanup
- def send_command

Data Fields

- **ccsds**
- **ip_address**
- **port**
- **command_socket**
- **endianness**
- **debug**

11.28.1 Detailed Description

```
The CommandInterface class provides methods to send CCSDS messages from the CFS test framework to CFS via
any app that listens on a UDP socket and injects CCSDS packets onto the software bus (TO or DIAG).
CommandInterface is a misnomer, as it is capable of sending both Command and Telemetry CCSDS packets.
```

11.28.2 Constructor & Destructor Documentation

11.28.2.1 def plugins.cfs.pycfs.command_interface.CommandInterface.__init__ (*self*, *ccsds*, *port* = `1234`, *ip* = `"127.0.0.1"`, *endianness* = `"little"`)

```
Constructor implementation for CommandInterface Class. It sets up the ip addr, port, ccsds version, etc.
```

11.28.3 Member Function Documentation

11.28.3.1 def plugins.cfs.pycfs.command_interface.CommandInterface.cleanup (*self*)

```
Performs requisite cleanup of the class, such as closing the socket.
@return None
```

11.28.3.2 def plugins.cfs.pycfs.command_interface.CommandInterface.init_socket (*self*)

```
Initialize socket connection.
@return None
```

11.28.3.3 def plugins.cfs.pycfs.command_interface.CommandInterface.send_command (*self*, *msg_id*, *function_code*, *data*, *header_args* = `None`)

```
This method constructs a CCSDS command packet and sends it
 to the ip:port defined when creating the class via UDP

@param msg_id: The message ID of the command to send
@param function_code: The app specific function/command code (CC)
@param data: A bytearray representing the packed message payload. This is specific to the message, so for now
the bytearray needs to be constructed by hand using struct.pack or the included BytePacker class
@param header_args: An optional dictionary of additional kwargs for the header constructor
 @return  The number of bytes that were sent over the socket. UDP is connectionless, so there is no way for the
socket to know that a packet was received by the destination
```

11.29 plugins.ccsds_plugin.readers.command_builder.CommandMessage Class Reference

Public Member Functions

- def **__init__**
- def **__getattr__**
- def **__setattr__**
- def **__delattr__**

Data Fields

- **command_codes**

11.29.1 Detailed Description

```
Class representing a CCSDS Command Message
```

11.30 plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def shutdown

Static Public Member Functions

- def control_flow_goto
- def if_condition
- def else_condition
- def end_condition
- def control_flow_conditional_goto
- def begin_loop
- def end_loop

Data Fields

- name

  Plugin Name.
- description

  Plugin Description.
- command_map

  Plugin Command Map.
- **begin_loop_index**

11.30.1 Detailed Description

```
The ControlFlow Plugin Class Definition

@note The Control-Flow Plugin provides the functionality of CTF control flow statement,
including looping and conditional statements.

@note The custom plugin class *must* inherit from the Plugin base-class.

@note All plugin functions mapped to a test instruction *must* return true/false to indicate pass/fail of
    that instruction.
```

11.30.2 Constructor & Destructor Documentation

11.30.2.1 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.__init__ (*self*)

```
Constructor of ControlFlow plugin.

@note The __init__ function is called once a plugin is loaded.

@note The __init__ function should not reference/interact with any other plugin since the other plugin may not
    be loaded at this stage.

@note The constructor of a plugin must define the following fields:
    - name
    - description
    - command map: dictionary mapping CTF instructions to a tuple defining the
          python function to use for that instruction, and a list of argument types
    - [optional] verify_required_commands: List of instructions that require verification (i.e polling
    until verification passes or timeout.
    - other class variables that can store state, etc...
```

11.30.3 Member Function Documentation

11.30.3.1 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.begin_loop (*label, conditions*) [static]

```
Create a loop entry point. The loop is identified by a unique label.
The BeginLoop must be in pairs with EndLoop instruction. The loop condition is defined in parameter
"conditions" as a list of variables and the associated comparison operations. The condition is True,
only if all comparison operations are True.

@param label: a user defined label (example: "LOOP_1")
@param conditions: a list of comparison conditions. Each includes "name", "operator" and "value".
 (example: {"name": "my_var", "operator": "<", "value": 20})

@return bool: always True .
```

11.30.3.2 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.control_flow_conditional_goto (*variable_name, operator, value, true_label = '', false_label = ''*) [static]

```
Deprecated function, may be removed in future.

@return bool: always True .
```

11.30.3.3 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.control_flow_goto (*command_index*) [static]

```
Deprecated function, may be removed in future.

@return bool: always True .
```

11.30.3.4 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.else_condition (*label*) [static]

```
Create a else conditional branch entry point. It must match a IfCondition and a EndCondition instruction
with the same label. It is optional in conditional branch block.
If the condition of IfCondition instruction is False, the control flow skips the 'if' branch block,
only executes the 'else' branch block. If ElseCondition instruction is not defined,
the control flow jumps to the end of conditional branch block defined by a EndCondition instruction.

@param label: a user defined label (example: "if_label_1")

@return bool: always True
```

11.30.3.5 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.end_condition (*label*) [static]

```
Create a if conditional branch exit point. It must match a IfCondition instruction with the same label.
When the control flow reaches EndCondition instruction, it exits the conditional branch block.

@param label: a user defined label (example: "if_label_1")

@return bool: always True
```

11.30.3.6 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.end_loop (*label*) [static]

```
       Create a loop exit point. It must match a BeginLoop instruction with the same label.
       If the looping condition in BeginLoop is False, the control flow jumps to the corresponding EndLoop instru
       and exits the loop.

@param label: a user defined label (example: "LOOP_1")

@return bool: always True
```

11.30.3.7 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.if_condition (*label, conditions*) [static]

```
Create a if conditional branch block entry point. It is identified by a unique label per test script.
The IfCondition must be in pairs with EndCondition instruction. ElseCondition instruction is optional.
The if condition is defined in parameter "conditions" as a list of variables
and the associated comparison operations. The condition is True, only if all comparison operations are True.

@param label: a user defined label (example: "if_label_1")
@param conditions: a list of comparison conditions. Each includes "name", "operator" and "value".
 (example: {"name": "my_var", "operator": "<", "value": 20})

@return bool: return True, unless conditions argument is not a list .
```

11.30.3.8 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.initialize (*self*)

```
Initialize implementation for the ControlFlow plugin.

@note The initialize function is called by the CTF plugin manager *after* all plugins have been loaded.

@note This function may interact with other plugins, since all plugins have been loaded at this stage.

@return bool: True if successful, False otherwise.
```

11.30.3.9 def plugins.control_flow_plugin.control_flow_plugin.ControlFlowPlugin.shutdown (*self*)

```
Shutdown implementation for the controlflow plugin.
@note The shutdown function is called by the CTF plugin manager upon completion of a test run.
@note The shutdown function can be exposed to test scripts by adding it to the command map.
```

11.31 lib.exceptions.CtfConditionError Class Reference

Public Member Functions

- def __init__

Data Fields

- **condition**

11.31.1 Detailed Description

```
CTF Condition Error thrown when a CTF Instruction Condition is not met during test run.
```

11.31.2 Constructor & Destructor Documentation

11.31.2.1 def lib.exceptions.CtfConditionError.__init__ (*self, message, test_condition*)

```
Constructor of CtfConditionError Class
```

11.32 lib.logger.CtfLogLevel Class Reference

Static Public Attributes

- int **TEST_PASS** = 21

- int **TEST_FAIL** = 22
- int **TEST_PASS_CONT** = 5
- int **TEST_FAIL_CONT** = 6

11.32.1 Detailed Description

```
CtfLogLevel: An enum containing custom log levels used in CTF
```

11.33 lib.exceptions.CtfParameterError Class Reference

Public Member Functions

- def __init__

Data Fields

- **parameter**

11.33.1 Detailed Description

```
CTF Parameter Error thrown when a CTF Instruction Parameter is invalid.
```

11.33.2 Constructor & Destructor Documentation

11.33.2.1 def lib.exceptions.CtfParameterError.__init__ (*self*, *message*, *parameter*)

```
Constructor of CtfParameterError Class
```

11.34 lib.exceptions.CtfTestError Class Reference

Public Member Functions

- def __init__

11.34.1 Detailed Description

```
General top-level exception that is thrown when a CTF Test Error occurs during a test run.
```

11.34.2 Constructor & Destructor Documentation

11.34.2.1 def lib.exceptions.CtfTestError.__init__ (*self*, *message*)

```
Constructor of CtfTestError Class
```

11.35 lib.ctf_global.CtfVerificationStage Class Reference

Static Public Attributes

- int **none** = 0
- int **first_ver** = 1
- int **polling** = 2
- int **last_ver** = 3

11.35.1 Detailed Description

```
Static class containing enumerations for verification stages of a CTF verification instruction.

@note The verification stage enums can be used to check which verification stage a CTF verification instruction i
on. Different logic can be implemented depending on the verification stage.
```

11.36 plugins.example_plugin.example_plugin.ExamplePlugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def test_verify_command
- def shutdown

Static Public Member Functions

- def test_command
- def test_shared_library

Data Fields

- name
  Plugin Name.
- description
  Plugin Description.
- command_map
  Plugin Command Map.
- verify_required_commands
  List of verification type commands.
- example_counter
  Counter to track how many verifications are ran.

11.36.1 Detailed Description

```
The Example Plugin Class Definition

@note The Example Plugin shows a simple CTF plugin that can perform a single test instruction and a single
    verification instruction, in addition to loading a C shared library.
```

```
@note The custom plugin class *must* inherit from the Plugin base-class.

@note A custom CTF plugin can be created to add new CTF instructions that can then be utilized within a JSON test
    script.

@note All plugin functions mapped to a test instruction *must* return true/false to indicate pass/fail of
    that instruction.
```

11.36.2 Constructor & Destructor Documentation

11.36.2.1 def plugins.example_plugin.example_plugin.ExamplePlugin.__init__ (*self*)

```
Constructor implementation for example plugin.

@note The __init__ function is called once a plugin is loaded.

@note The __init__ function should not reference/interact with any other plugin since the other plugin may not
    be loaded at this stage.

@note The constructor of a plugin must define the following fields:
    - name
    - description
    - command map: dictionary mapping CTF instructions to a tuple defining the
          python function to use for that instruction, and a list of argument types
    - [optional] verify_required_commands: List of instructions that require verification (i.e polling
    until verification passes or timeout.
    - other class variables that can store state, etc...
```

11.36.3 Member Function Documentation

11.36.3.1 def plugins.example_plugin.example_plugin.ExamplePlugin.initialize (*self*)

```
Initialize implementation for the example plugin.

@note The initialize function is called by the CTF plugin manager *after* all plugins have been loaded.

@note This function may interact with other plugins, since all plugins have been loaded at this stage.

@return bool: True if successful, False otherwise.
```

11.36.3.2 def plugins.example_plugin.example_plugin.ExamplePlugin.shutdown (*self*)

```
Shutdown implementation for the example plugin.
@note The shutdown function is called by the CTF plugin manager upon completion of a test run.
@note The shutdown function can be exposed to test scripts by adding it to the command map.
```

11.36.3.3 def plugins.example_plugin.example_plugin.ExamplePlugin.test_command (*arg1*, *arg2*) [static]

```
Simply logs that the test command was executed with the provided arguments.

@param arg1: any value (example: "Hello")
@param arg2: any value (example: "World")

@return bool: True if successful, False otherwise.
```

11.36.3.4 def plugins.example_plugin.example_plugin.ExamplePlugin.test_shared_library ( ) [static]

```
Uses libc to get the system time and log it to system output.

@note Verifies that the expected number of bytes were printed.
```

```
@return bool: True if successful, False otherwise.
```

11.36.3.5 def plugins.example_plugin.example_plugin.ExamplePlugin.test_verify_command (*self*)

```
Increments the plugin's example_counter value and checks if it is greater than `5`.

@note Verification instructions will be re-executed by the CTF core until the verification passes, or the
    verification timeout is reached.

@return bool: True if successful, False otherwise.
```

11.36.4 Field Documentation

11.36.4.1 plugins.example_plugin.example_plugin.ExamplePlugin.example_counter

Counter to track how many verifications are ran.

Other plugin-specific properties can also be defined

11.37 lib.ftp_interface.FtpInterface Class Reference

Public Member Functions

- def __init__
- def store_file_ftp
- def get_file_ftp
- def upload_ftp
- def download_ftp
- def connect_ftp
- def disconnect_ftp
- def upload_ftputil
- def download_ftputil

Data Fields

- **uploadlevel**
- **ftp**
- **curdir**
- **ipaddr**
- **ftpconnect**
- **ftp_timeout**
- **remotebase**

11.37.1 Detailed Description

```
The FtpInterface class provides functionality to connect/disconnect to remote FTP server,
upload/download files, create folder on server.
@note - Two parallel FTP implementations are provided: ftputil for use via SSH, and ftplib for SP0
```

11.37.2 Constructor & Destructor Documentation

11.37.2.1 def lib.ftp_interface.FtpInterface.__init__ (*self*)

```
 Constructor for FtpInterface class. Set default values for FtpInterface attributes,
 such as ipaddr, ftp_timeout, etc.
```

11.37.3 Member Function Documentation

11.37.3.1 def lib.ftp_interface.FtpInterface.connect_ftp (*self, ipaddr, usrid*)

```
Connect to FTP server, and set the FtpInterface attributes.
@param ipaddr: the IP address of FTP server.
@param usrid: the user id to connect to the FTP server.
@return bool: True if successfully connect to FTP server, False otherwise.
```

11.37.3.2 def lib.ftp_interface.FtpInterface.disconnect_ftp (*self*)

```
Disconnect to FTP server, and reset the FtpInterface attributes.
@return None
```

11.37.3.3 def lib.ftp_interface.FtpInterface.download_ftp (*self, remotepath, ipaddr =* None, *localpath =* None, *file =* None, *usr_id =* None)

```
Download a file or files from the FTP server to the local computer.
@param remotepath: the path to the download file/files on the FTP server.
@param ipaddr: the IP address of FTP server. If it is None, use the previous FTP connection,
        otherwise re-connect FTP server using ipaddr and usr_id.
@param localpath: the path to store the downloaded file/files on local computer.
@param file: the file to be downloaded from the FTP server. If the file is None,
     all files in remotepath will be downloaded.
@param usr_id: the user id to connect to the FTP server.
@return bool: True if download successfully, False otherwise.
```

11.37.3.4 def lib.ftp_interface.FtpInterface.download_ftputil (*self, host, remote_path, local_path, usrid =* 'anonymous')

```
FTP download utility: download a whole folder content from the FTP host to the local computer.
@param host: FTP server host/IP.
@param remote_path: the FTP server path.
@param local_path: the local computer path to store downloaded files.
@param usrid: the user id to connect to the FTP server. The default user is anonymous'.
@return bool: True if download successfully, False otherwise.
```

11.37.3.5 def lib.ftp_interface.FtpInterface.get_file_ftp (*self, remote_file, local_path =* None)

```
Download a file from the FTP server to the local computer.
@param remote_file: the path/name of the file on FTP server.
@param local_path: the path to store the transferred file on local computer.
@return bool: True if the file is downloaded successfully, False otherwise.
```

11.37.3.6 def lib.ftp_interface.FtpInterface.store_file_ftp (*self, path, file*)

```
Transfer file to FTP server using the FTP command STOR. The file transfer is in binary mode.
@param path: the path of the transfer file on local computer.
@param file: the name of the transfer file on local computer.
@return bool: True if the file is transferred successfully, False otherwise.
```

11.37.3.7 def lib.ftp_interface.FtpInterface.upload_ftp (*self*, *localpath*, *ipaddr* = None, *remotepath* = None, *file* = None, *usr_id* = None)

```
Upload a file or files from the local computer to the FTP server.
@param localpath: the path of the uploaded file/files on local computer.
@param ipaddr: the IP address of FTP server. If it is None, use the previous FTP connection,
        otherwise re-connect FTP server using ipaddr and usr_id.
@param remotepath: the path to store the uploaded file/files on the FTP server.
@param file: the file to be uploaded on local computer. If the file is None,
     all files in localpath will be uploaded.
@param usr_id: the user id to connect to the FTP server.
@return bool: True if upload successfully, False otherwise.
```

11.37.3.8 def lib.ftp_interface.FtpInterface.upload_ftputil (*self*, *host*, *local_path*, *remote_path*, *usrid* = 'anonymous')

```
FTP upload utility: upload a whole folder content from the local computer to the FTP host.
@param host: FTP server host/IP.
@param local_path: the local computer path.
@param remote_path: the FTP server path to store the uploaded files.
@param usrid: the user id to connect to the FTP server. The default user is anonymous'
@return bool: True if upload successfully, False otherwise.
```

11.38 lib.ctf_global.Global Class Reference

Static Public Member Functions

- def create_arg_parser
- def load_config
- def set_time_manager
- def get_time_manager

Static Public Attributes

- config = None

  Config parser for the designated config file, initialized in load_config.
- tuple plugins_available = dict()

  Dictionary of loaded plugins.
- plugin_manager = None

  Reference to the plugin manager object.
- string current_script_log_dir = ""

  Log directory of current script.
- string test_log_dir = ""

  Log directory of the complete test run (includes log directory of scripts)
- string CTF_log_dir = ""

  Temporary logging directory for CTF.
- CTF_log_dir_file = None

  CTF top-level log file.
- time_manager = None

  Current time manager used by CTF.
- test_start_time = None

  Start time of current test run.
- current_verification_start_time = None

Start time of current verification.

- current_verification_stage = CtfVerificationStage.none

  Current verification stage.
- current_instruction_index = None

  [Read-Only] Current Instruction Index, default value is None.
- goto_instruction_index = None

  [Read-Only] Current goto instruction index, default value is None.
- dictionary variable_store = {}

  [Read-Only] Variable Storage.
- dictionary **label_map** = {}
- dictionary **goto_label_map** = {}
- dictionary **conditional_branch_map** = {}

11.38.1 Detailed Description

```
Static class containing globally accessible CTF and plugin data.
```

11.38.2 Member Function Documentation

11.38.2.1 def lib.ctf_global.Global.create_arg_parser ( ) [static]

```
Creates and returns an argument parser for command line args.
```

11.38.2.2 def lib.ctf_global.Global.get_time_manager ( ) [static]

```
Gets the currently active time manager
```

11.38.2.3 def lib.ctf_global.Global.load_config (*config_file*) [static]

```
Loads the config file specified and sets the workspace_dir environment variable

@note - Command line arguments are not visible here, so the status message indicates if the default config
is being used in case it was not explicitly provided.
@note - If the config file does not exist, the application will exit with an error.
@note - The config field cfs:workspace_dir will be set as an environment variable for the current process.

@return str: An optional status message, since logging will not have been configured yet
```

11.38.2.4 def lib.ctf_global.Global.set_time_manager (*time_manager*) [static]

```
Sets the currently active time manager.

@note - A custom plugin time manager *must* inherit from the TimeManager class and implement its methods
```

11.38.3 Field Documentation

11.38.3.1 string lib.ctf_global.Global.CTF_log_dir = "" [static]

Temporary logging directory for CTF.

Contents of the temporary directory are moved to the test log directory on test completion.

11.38.3.2 lib.ctf_global.Global.CTF_log_dir_file = None [static]

CTF top-level log file.

Includes CTF core logs such as initialization and plugin loading/unloading

11.38.3.3 lib.ctf_global.Global.current_instruction_index = None [static]

[Read-Only] Current Instruction Index, default value is None.

current_instruction_index is updated by lib/test.py to track the execution instruction index of the test. Use Utility function "get_current_instruction_index()" to get the index value (int).

11.38.3.4 string lib.ctf_global.Global.current_script_log_dir = "" [static]

Log directory of current script.

Useful when needing to write data to the current log directory.

11.38.3.5 lib.ctf_global.Global.current_verification_stage = CtfVerificationStage.none [static]

Current verification stage.

Use CtfVerificationStage to evaluate what verification stage CTF is currently at.

11.38.3.6 lib.ctf_global.Global.goto_instruction_index = None [static]

[Read-Only] Current goto instruction index, default value is None.

Control Flow Plugins can set the next instruction index to execute based on user input or logic within the plugin. Do not use it directly

11.38.3.7 lib.ctf_global.Global.plugin_manager = None [static]

Reference to the plugin manager object.

May be used to invoke instructions (or access) other plugins.

11.38.3.8 tuple lib.ctf_global.Global.plugins_available = dict() [static]

Dictionary of loaded plugins.

Set by the CTF core after loading plugins.

11.38.3.9 lib.ctf_global.Global.time_manager = None [static]

Current time manager used by CTF.

Utilized by other plugins to manage time

11.38.3.10 dictionary lib.ctf_global.Global.variable_store = {} [static]

[Read-Only] Variable Storage.

Recommend using utility functions to set/get variables.

11.39 lib.event_types.Instruction Class Reference

Public Member Functions

- def **__init__**

Data Fields

- **delay**
- **command**
- **test**
- **command_index**
- **is_disabled**

11.39.1 Detailed Description

```
Represents a single CTF Test Instruction.

@param delay: The time in seconds to wait before executing this instruction
@param command: The dict containing instruction parameters
@param test: Integer index of the test case that includes this instruction
@param command_index: Integer index of this instruction within the test case
@param disabled: Whether or not the instruction is disabled
```

11.40 lib.readers.json_script_reader.JSONScriptReader Class Reference

Public Member Functions

- def __init__
- def process_header
- def process_functions
- def sanitize_args
- def process_tests
- def resolve_function
- def **resolve_function_params**

Data Fields

- **raw_data**
- **valid_script**
- **script**
- **input_script_path**
- **functions**

11.40.1 Detailed Description

```
The JSONScriptReader class provides methods to parse a CTF JSON test script.

@param input_script_path: The path to the input JSON script
```

11.40.2 Constructor & Destructor Documentation

11.40.2.1 def lib.readers.json_script_reader.JSONScriptReader.__init__ (*self*, *input_script_path*)

```
Constructor for the JSONScriptReader class.
Loads and parses the contents of a single JSON test script file, and resolves imports
```

11.40.3 Member Function Documentation

11.40.3.1 def lib.readers.json_script_reader.JSONScriptReader.process_functions (*self*)

```
Parse the function definitions and imports in the test script
```

11.40.3.2 def lib.readers.json_script_reader.JSONScriptReader.process_header (*self*)

```
Parse and process test information from script header
```

11.40.3.3 def lib.readers.json_script_reader.JSONScriptReader.process_tests (*self*)

```
Iterates over test cases within the test script and parses each test case.
```

11.40.3.4 def lib.readers.json_script_reader.JSONScriptReader.resolve_function (*self*, *name*, *params*, *functions*)

```
Perform in-line replacement of function calls with the set of instructions within the function definition
```

11.40.3.5 def lib.readers.json_script_reader.JSONScriptReader.sanitize_args (*self*, *args*)

```
Iterates over arguments within test instructions and decodes arguments if needed.
```

11.41 plugins.cfs.pycfs.local_cfs_interface.LocalCfsInterface Class Reference

Public Member Functions

- def __init__
- def get_start_string
- def build_cfs
- def start_cfs

Data Fields

- **init_passed**
- **cfs_std_out_path**

Additional Inherited Members

11.41.1 Detailed Description

```
Lower-level interface to communicate with cFS locally (linux)
```

11.41.2 Constructor & Destructor Documentation

11.41.2.1 def plugins.cfs.pycfs.local_cfs_interface.LocalCfsInterface.__init__ (*self*, *config*, *telemetry*, *command*, *mid_map*, *ccsds*)

```
Constructor implementation for LocalCfsInterface Class.
if configured to build cfs, build cfs. otherwise set init_passed to True
```

11.41.3 Member Function Documentation

11.41.3.1 def plugins.cfs.pycfs.local_cfs_interface.LocalCfsInterface.build_cfs (*self*)

```
Build cfs image. The path of cFS source is configured in config init file.
The build output folder is also configured in init file.
@return bool: True if build succeed, otherwise False
```

11.41.3.2 def plugins.cfs.pycfs.local_cfs_interface.LocalCfsInterface.get_start_string (*self*, *run_args*)

```
Get the command string/path to start cfs (linux)
@param run_args: run_time argument to start cfs
@return String: full command string to start cfs
```

11.41.3.3 def plugins.cfs.pycfs.local_cfs_interface.LocalCfsInterface.start_cfs (*self*, *run_args*)

```
Start the cfs instance process.
@param run_args: run_args is used to build the start_string.
@return dictionary: the return result_values is a dictionary, including 'results': True if cfs instance
starts successfully, otherwise False;  and 'pid': the pid of cfs instance process.
```

11.42 lib.status.ObjectFactory Class Reference

Static Public Member Functions

- def **create_object**

Static Private Member Functions

- def **__create_suite_status**
- def **__create_test_status**
- def **__create_instruction_status**
- def **__create_script_status**
- def **__create_plugin_info**
- def **__create_command_info**
- def **__create_parameter_info**

11.42.1 Detailed Description

```
This class defines enumerations for the status definitions used by CTF to send instruction status.
```

11.43 plugins.cfs.pycfs.output_app_interface.OutputManager Class Reference

Public Member Functions

- def __init__
- def enable_output
- def disable_output

Data Fields

- **local_ip**
- **local_port**
- **command_interface**
- **ccsds_ver**
- **command_args**
- **command_mids**

11.43.1 Detailed Description

```
Base class that each output application must inherit from.
within this class, define the methods that all of the output applications must implement
```

11.43.2 Constructor & Destructor Documentation

11.43.2.1 def plugins.cfs.pycfs.output_app_interface.OutputManager.__init__ (*self*, *local_ip*, *local_port*, *command_interface*, *ccsds_ver*, *command_mids* = None)

```
Constructor implementation for OutputManager class. It sets up the local_ip, local_port, command_interface,
ccsds version, command_args, command_mids.
```

11.43.3 Member Function Documentation

11.43.3.1 def plugins.cfs.pycfs.output_app_interface.OutputManager.disable_output (*self*)

```
Define abstract disable_output method, the inherited class must implement
```

11.43.3.2 def plugins.cfs.pycfs.output_app_interface.OutputManager.enable_output (*self*)

```
Define abstract enable_output method, the inherited class must implement
```

11.44 lib.plugin_manager.Plugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def process_command
- def shutdown

Data Fields

- name

    Plugin Name.
- description

    Plugin Description.
- command_map

    Plugin Command Map.
- verify_required_commands

    List of verification type instructions.
- continuous_verification_commands

    List of continuously verified instructions (i.e executed every poll without an explicit instruction)
- end_test_on_fail_commands

    List of instructions that end test on failure (i.e critical instructions that the test script cannot proceed without)

11.44.1 Detailed Description

```
Base class that each plugin must inherit from. This class defines methods and properties that all plugins may
override or implement.
```

11.44.2 Constructor & Destructor Documentation

11.44.2.1 def lib.plugin_manager.Plugin.__init__ (*self*)

```
Constructor of Plugin Class: Initiate instance properties
```

11.44.3 Member Function Documentation

11.44.3.1 def lib.plugin_manager.Plugin.initialize (*self*)

```
Virtual initialize method definition. Must be overridden by child Plugin class.
@note - The initialize method is called for each plugin after *all* plugins are loaded.
```

11.44.3.2 def lib.plugin_manager.Plugin.process_command (*self*, *kwargs*)

```
Given a CTF Test Instruction, this function finds the first plugin that "contains" that test instruction within
its command map. Once a valid plugin is found, the implementation of that instruction is invoked using
keyworded variable length of arguments in kwargs.

@note - This function will ensure that the number of argument provided to the plugin's function is greater than
the number of required arguments (non-optional), and less than or equal to the total number of arguments
(required + optional)
```

11.44.3.3 def lib.plugin_manager.Plugin.shutdown (*self*)

```
Virtual shutdown method definition. Must be overridden by child Plugin class.
@note - The shutdown method is called for each plugin after test execution is complete. Use this function to
shutdown/cleanup any external interfaces or data.
```

11.44.4 Field Documentation

11.44.4.1 lib.plugin_manager.Plugin.command_map

Plugin Command Map.

The command map utilizes the instruction name as the key, with the value being a tuple of instruction implementation and argument types.

Note

Example: {"TestCommand": (self.test_command, [ArgTypes.string] ∗ 2)}

11.45 lib.plugin_manager.PluginManager Class Reference

Public Member Functions

- def __init__
- def initialize_plugins
- def shutdown_plugins
- def find_plugin_for_command
- def find_plugin_for_command_and_execute
- def reload_plugins
- def walk_package
- def create_plugin_info

Data Fields

- **plugin_packages**
- **plugins**
- **plugin_name_list**
- **seen_paths**
- **disabled_plugins**

11.45.1 Detailed Description

```
Upon creation, this class will read the plugins package for modules
that contain a class definition that is inheriting from the Plugin class
```

11.45.2 Constructor & Destructor Documentation

11.45.2.1 def lib.plugin_manager.PluginManager.__init__ (*self*, *plugin_packages*)

```
Constructor of PluginManager Class: initiates the reading of all available plugins
when an instance of the PluginManager object is created
```

11.45.3 Member Function Documentation

11.45.3.1 def lib.plugin_manager.PluginManager.create_plugin_info (*self*, *directory*)

```
Outputs the plugin information files in JSON format for utilization by the CTF editor or other tools.

@param directory - Directory to write the plugin information files.
@note - The directory is created automatically if it does not exist.
```

11.45.3.2 def lib.plugin_manager.PluginManager.find_plugin_for_command (*self*, *command*)

```
Given a CTF Test Instruction, find the plugin instance that can execute that instruction.

@note - CTF Test Instructions must be named uniquely across different plugins.
@note - It is recommended to prefix the instruction name with a plugin identifier to avoid ambiguity. For
example: MyPlugin_DoSomething

@return Plugin: Plugin instance found that implements the given instruction. None of no plugins found.
```

11.45.3.3 def lib.plugin_manager.PluginManager.find_plugin_for_command_and_execute (*self*, *command*)

```
Given a CTF Test Instruction, find the plugin instance that can execute that instruction, execute the
instruction and return the instruction status (pass/fail)

@return Plugin: Boolean: CTF Instruction Status (True/False)
```

11.45.3.4 def lib.plugin_manager.PluginManager.initialize_plugins (*self*)

```
After loading all plugins, this function calls initialize() on all loaded plugins within the plugin manager
```

11.45.3.5 def lib.plugin_manager.PluginManager.reload_plugins (*self*)

```
Reset the list of all plugins and initiate the walk over the main
provided plugin package to load all available plugins
```

11.45.3.6 def lib.plugin_manager.PluginManager.shutdown_plugins (*self*)

```
Before CTF shutdown (or on plugin restart), this function calls shutdown() on all loaded plugins within
the plugin manager
```

11.45.3.7 def lib.plugin_manager.PluginManager.walk_package (*self*, *package*)

```
Recursively walk the supplied package to retrieve all plugins

@param package - Given a package path, this function recursively walks through the package and imports any
        modules available within the package.
```

11.46 plugins.cfs.cfs_config.RemoteCfsConfig Class Reference

Public Member Functions

- def __init__
- def load_config_data

Data Fields

- **destination**
- **cfs_protocol**
- **cfs_run_in_xterm**

11.46.1 Detailed Description

```
CFS Configuration for SSH targets, inherited from CfsConfig class.
```

11.46.2 Constructor & Destructor Documentation

11.46.2.1 def plugins.cfs.cfs_config.RemoteCfsConfig.__init__ (*self, name*)

```
Constructor for RemoteCfsConfig Class. Override cfs_protocol attribute to ssh.
```

11.46.3 Member Function Documentation

11.46.3.1 def plugins.cfs.cfs_config.RemoteCfsConfig.load_config_data (*self, section_name*)

```
From loaded sections of INI config, interpret CFS target config attributes, including
build_cfs, CCSDS_data_dir, CCSDS_target, etc.
@param section_name: loaded Json CFS target section.
@return None
```

11.47 plugins.cfs.pycfs.cfs_controllers.RemoteCfsController Class Reference

Public Member Functions

- def __init__
- def initialize
- def archive_cfs_files
- def shutdown_cfs
- def shutdown

Data Fields

- **execution**
- **cfs**
- **cfs_process_list**
- **cfs_running**

Additional Inherited Members

11.47.1 Detailed Description

```
RemoteCfsController class Definition:

@note RemoteCfsController class is inherited from CfsController class. It only redefines a few functions,
      including __init__, initialize, archive_cfs_files, shutdown_cfs, shutdown.
@note RemoteCfsController is initiated when INI config file uses 'ssh' protocol.
```

11.47.2 Constructor & Destructor Documentation

11.47.2.1 def plugins.cfs.pycfs.cfs_controllers.RemoteCfsController.__init__ (*self, config*)

```
Constructor implementation for RemoteCfsController class.
```

11.47.3 Member Function Documentation

11.47.3.1 def plugins.cfs.pycfs.cfs_controllers.RemoteCfsController.archive_cfs_files (*self*, *source_path*)

```
Implementation of CFS plugin instructions archive_cfs_files. When CFS plugin instructions
(archive_cfs_files) is executed, it calls RemoteCfsController instance’s archive_cfs_files function.
```

11.47.3.2 def plugins.cfs.pycfs.cfs_controllers.RemoteCfsController.initialize (*self*)

```
Initialize CfsController instance, including the followings: create mid map; import ccsds header;
create ssh CFS command interface; create telemetry interface;
```

11.47.3.3 def plugins.cfs.pycfs.cfs_controllers.RemoteCfsController.shutdown (*self*)

```
This function will shut down the CFS application being tested even if the JSON test file does not
include the shutdown test command
```

11.47.3.4 def plugins.cfs.pycfs.cfs_controllers.RemoteCfsController.shutdown_cfs (*self*)

```
Implementation of CFS plugin instructions shutdown_cfs. When CFS plugin instructions
(shutdown_cfs) is executed, it calls RemoteCfsController instance’s shutdown_cfs function.
```

11.48 plugins.cfs.pycfs.remote_cfs_interface.RemoteCfsInterface Class Reference

Public Member Functions

- def __init__
- def get_start_string
- def start_cfs
- def build_cfs

Data Fields

- **execution_controller**
- **cfs_std_out_path**

Additional Inherited Members

11.48.1 Detailed Description

```
RemoteCfsInterface implements lower-level interface to communicate with cFS remotely over SSH.
Inherits Cfs Interface - extends some of it’s functionality specifically for SSH.
```

11.48.2 Constructor & Destructor Documentation

11.48.2.1 def plugins.cfs.pycfs.remote_cfs_interface.RemoteCfsInterface.__init__ (*self*, *config*, *telemetry*, *command*, *mid_map*, *ccsds*, *execution*)

```
Constructor implementation for RemoteCfsInterface.
Pass arguments to base class.
```

11.48.3 Member Function Documentation

11.48.3.1 def plugins.cfs.pycfs.remote_cfs_interface.RemoteCfsInterface.build_cfs (*self*)

```
Build remote cfs image. The path of cFS source is configured in config init file.
The build output folder is also configured in init file.
@return bool: True if build succeed, otherwise False
```

11.48.3.2 def plugins.cfs.pycfs.remote_cfs_interface.RemoteCfsInterface.get_start_string (*self*, *run_args*)

```
Build the start string for starting cFS instance.
@param run_args: run_args is used to build start string
@return string: command to start cFS instance, including remote cFS path.
```

11.48.3.3 def plugins.cfs.pycfs.remote_cfs_interface.RemoteCfsInterface.start_cfs (*self*, *run_args*)

```
 Start the remote cfs instance process.
 @param run_args: run_args is used to build the start_string.
 @return dictionary: the return result_values is a dictionary, including 'results': True if cfs instance
 starts successfully, otherwise False;  and 'pid': the pid of cfs instance process.
```

11.49 lib.script_manager.ScriptManager Class Reference

Public Member Functions

- def **__init__**
- def add_script
- def add_script_file
- def run_all_scripts
- def prep_logging
- def write_summary_line
- def __del__

Data Fields

- **script_list**
- **config**
- **regression_summary_file_path**
- **regression_summary_json_file_path**
- **curr_script_log_dir_path**
- **plugin_manager**
- **status_manager**
- **summary_file**

11.49.1 Detailed Description

```
The ScriptManager class adds and manages all loaded CTF test scripts.
@note - The script manager's add_script is called with each script loaded by the JSONScriptReader.
@note - The script manager handles execution of test scripts, including logging the results and managing the
test suite status

@param plugin_manager: Initialized instance of the plugin manager, used to interact with the loaded plugins
@param status_manager: Initialized instance of the status manager, used to send status to external listeners
```

11.49.2 Constructor & Destructor Documentation

11.49.2.1 def lib.script_manager.ScriptManager.__del__ (*self*)

```
Destructor implementation to close summary file on deletion of the ScriptManager
```

11.49.3 Member Function Documentation

11.49.3.1 def lib.script_manager.ScriptManager.add_script (*self, script*)

```
Adds a script to the list of scripts managed by the script manager
```

11.49.3.2 def lib.script_manager.ScriptManager.add_script_file (*self, file*)

```
Adds a script file to the list of scripts. If the file is not valid, skip it.
```

11.49.3.3 def lib.script_manager.ScriptManager.prep_logging (*self*)

```
Prepares logging directories for a CTF test run. Logging directories will include script-specific log
directories, as well as high-level log files and results summary.
```

11.49.3.4 def lib.script_manager.ScriptManager.run_all_scripts (*self*)

```
Run all added scripts, updating the status packets, and ensuring plugins are reloaded between scripts if needed.
```

11.49.3.5 def lib.script_manager.ScriptManager.write_summary_line (*self, summary_line*)

```
Write an entry to the summary results file(s).
@note - An entry consists of:
- Script status (pass/fail)
- Execution Time
- Verification Number
- Requirements Verified
- # of tests that ran
- # of tests that passed
- # of tests the failed
- # of tests with an error
- Script input file (.JSON)
```

11.50 lib.script_manager.ScriptManagerConfig Class Reference

Public Member Functions

- def __init__

Data Fields

- **reset_plugins_between_scripts**
- **json_results**

11.50.1 Detailed Description

```
Configuration parameters used by the ScriptManager class, obtained from the loaded INI config
```

11.50.2 Constructor & Destructor Documentation

11.50.2.1 def lib.script_manager.ScriptManagerConfig.__init__ (*self*)

```
Constructor of ScriptManagerConfig class. Initialize properties from INI file
```

11.51 plugins.ssh.ssh_plugin.SshConfig Class Reference

Public Member Functions

- def __init__

Data Fields

- command_timeout
  SshConfig command_timeout property.
- print_stdout
  SshConfig print_stdout property.
- log_stdout
  SshConfig log_stdout property.

11.51.1 Detailed Description

```
The SshConfig helper Class Definition

@note it gets the command_timeout, print_stdout and print_stdout from configuration Json file
```

11.51.2 Constructor & Destructor Documentation

11.51.2.1 def plugins.ssh.ssh_plugin.SshConfig.__init__ (*self*)

```
Constructor implementation for SshConfig helper class.
```

11.52 plugins.ssh.ssh_plugin.SshController Class Reference

Public Member Functions

- def __init__
- def init_connection
- def run_command
- def run_command_persistent
- def get_last_pid
- def run_command_local
- def check_output
- def put_file
- def get_file
- def rsync
- def upload_ftp
- def download_ftp
- def shutdown

Data Fields

- **config**
- **connection**
- **last_result**
- **last_pid**
- **ftp_interface**

11.52.1 Detailed Description

```
The SshController helper Class Definition

@note SshController provides an instance of SSH plugin's target:  self.targets[name] = SshController(SshConfig())

@note SshController provides the implementation SSH plugin's commands. For example, upload_ftp commands calls
      self.targets[name].download_ftp(host, remote_path, local_path)
```

11.52.2 Constructor & Destructor Documentation

11.52.2.1 def plugins.ssh.ssh_plugin.SshController.__init__ (*self*, *config*)

```
Constructor implementation for SshController helper class.
```

11.52.3 Member Function Documentation

11.52.3.1 def plugins.ssh.ssh_plugin.SshController.check_output (*self*, *output_contains* = None, *output_does_not_contain* = None, *exit_code* = 0)

```
check_output provides implementation of SSH plugin's check_output / SSH_CheckOutput method:
self.targets[name].check_output(output_contains, output_does_not_contain, exit_code)
```

11.52.3.2 def plugins.ssh.ssh_plugin.SshController.download_ftp (*self*, *host*, *remote_path*, *local_path*)

```
  download_ftp provides implementation of SSH plugin's download_ftp / SSH_GetFTP method:
self.targets[name].download_ftp(host, remote_path, local_path)
```

11.52.3.3 def plugins.ssh.ssh_plugin.SshController.get_file (*self*, *remote_path*, *local_path*, *args* = None)

```
  get_file provides implementation of SSH plugin's get_file / SSH_GetFile method:
self.targets[name].get_file(remote_path, local_path, args)
```

11.52.3.4 def plugins.ssh.ssh_plugin.SshController.get_last_pid (*self*)

```
  return  last_pid
```

11.52.3.5 def plugins.ssh.ssh_plugin.SshController.init_connection (*self*, *host*, *user* = None, *port* = None, *gateway* = None, *ssh_config_path* = None, *args* = None)

```
init_connection provides implementation of SSH plugin's init_connection method:
self.targets[name].init_connection(host, user, port, gateway, ssh_config_path, args)
```

11.52.3.6 def plugins.ssh.ssh_plugin.SshController.put_file (*self*, *local_path*, *remote_path*, *args* = None)

```
 put_file provides implementation of SSH plugin's put_file / SSH_PutFile method:
self.targets[name].put_file(local_path, remote_path, args)
```

11.52.3.7 def plugins.ssh.ssh_plugin.SshController.rsync (*self*, *source*, *dest*, *push*, *args* = None)

```
  rsync implements async file transfer
```

11.52.3.8 def plugins.ssh.ssh_plugin.SshController.run_command (*self*, *command*, *cwd* = "", *prefix* = ":")

```
run_command provides implementation of SSH plugin's run_command / SSH_RunRemoteCommand method:
self.targets[name].run_command(command, cwd, prefix)
```

11.52.3.9 def plugins.ssh.ssh_plugin.SshController.run_command_local (*self*, *command*)

```
run_command_local provides implementation of SSH plugin's run_command_local / SSH_RunLocalCommand method:
self.targets[name].run_command_local(command)
```

11.52.3.10 def plugins.ssh.ssh_plugin.SshController.run_command_persistent (*self*, *command*, *cwd* = "", *prefix* = ":")

```
run_command_persistent implement SSH persistent call with configurable time-out
```

11.52.3.11 def plugins.ssh.ssh_plugin.SshController.shutdown (*self*)

```
   shutdown provides implementation of SSH plugin's shutdown method:
   SSH plugin calls self.targets[name].shutdown()
```

11.52.3.12 def plugins.ssh.ssh_plugin.SshController.upload_ftp (*self*, *host*, *local_path*, *remote_path*)

```
  upload_ftp provides implementation of SSH plugin's upload_ftp / SSH_PutFTP method:
self.targets[name].upload_ftp(host, local_path, remote_path)
```

11.53 plugins.ssh.ssh_plugin.SshPlugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def register_target
- def init_connection
- def run_command
- def run_command_local
- def check_output
- def put_file
- def get_file
- def upload_ftp
- def download_ftp
- def shutdown

Data Fields

- **name**
- **description**
- **targets**
- **command_map**
- **verify_required_commands**

11.53.1 Detailed Description

```
The SSH Plugin Class Definition

@note The SSH Plugin provides remote and local shell command execution capability for CTF.

@note The following test instructions are available:

@note SSH_RegisterTarget;SSH_InitSSH; SSH_RunRemoteCommand;SSH_RunLocalCommand; SSH_CheckOutput; SSH_PutFile;

@note SSH_GetFile; SSH_GetFTP; SSH_PutFTP;

@note A custom CTF plugin can be created to add new CTF instructions that can then be utilized within a JSON test
    script.

@note All plugin functions mapped to a test instruction *must* return true/false to indicate pass/fail of
    that instruction.
```

11.53.2 Constructor & Destructor Documentation

11.53.2.1 def plugins.ssh.ssh_plugin.SshPlugin.__init__ (*self*)

```
Constructor implementation for SSH plugin.

@note The __init__ function is called once a plugin is loaded.

@note The __init__ function should not reference/interact with any other plugin since the other plugin may not
    be loaded at this stage.

@note The constructor of a plugin must define the following fields:
    - name
    - description
    - command map: dictionary mapping CTF instructions to a tuple defining the
          python function to use for that instruction, and a list of argument types
    - [optional] verify_required_commands: List of instructions that require verification (i.e polling
    until verification passes or timeout.
    - other class variables that can store state, etc...
```

11.53.3 Member Function Documentation

11.53.3.1 def plugins.ssh.ssh_plugin.SshPlugin.check_output (*self*, *output_contains* = None, *output_does_not_contain* = None, *exit_code* = 0, *name* = "default")

```
Compares the output of the most recently executed command.
ExecutionRunRemoteCommand or ExecutionRunLocalCommand must be called first.

@param name: A name already registered with SSH_RegisterTarget to identify the connection. (Optional)
@param output_contains: A substring that must be contained in stdout. (Example: "PASS") (Optional)
@param output_does_not_contain: A substring that should not be contained in stdout. (Example: "FAIL") (Optional)
@param exit_code: The expected exit code after the shell command is executed. (Optional default = 0)

@return bool: True if successful, False otherwise.
```

```
@par Example:
@code
{
    "command": "SSH_CheckOutput",
    "wait": 0,
    "data": {
"name": "workstation",
"output_contains": "Built target mission-install",
"output_does_not_contain": "Error",
"exit_code": 0
    }
}
```

11.53.3.2 def plugins.ssh.ssh_plugin.SshPlugin.download_ftp (*self*, *host*, *remote_path*, *local_path*, *name* = "default")

```
Downloads a path (file or directory) from the FTP server to the local filesystem.

@param name: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
@param host: The hostname or address of the FTP server.
@param remote_path: The path to the source file or directory on the FTP server.
@param local_path: The local path to where the file or directory is to be downloaded.

@return bool: True if successful, False otherwise.

@par Example:
@code
{
    "command": "SSH_GetFTP",
    "wait": 0,
    "data": {
"name": "workstation",
"host": "ftphost",
"remote_path": "./data/output.dat",
"local_path": "./results.txt"
    }
}
```

11.53.3.3 def plugins.ssh.ssh_plugin.SshPlugin.get_file (*self*, *remote_path*, *local_path*, *args* = None, *name* = "default")

```
Copies a path (file or directory) from the remote host to the local filesystem via rsync.
Relative or absolute paths are allowed, but do not use ~. Strings are passed directly to rsync,
so the same rules apply regarding paths, patterns, etc.

@param name: A name already registered with SSH_RegisterTarget to identify the connection. (Optional)
@param remote_path: The path to where the file or directory is to be copied.
            For remote hosts use the SSH syntax user@host:path.
@param local_path: The path to the local file or directory to be copied.

@param args: An object that describes optional parameters for the transfer.
     delete: A boolean corresponding to rsync's --delete option.
     If true, rsync will remove remote files that no longer exist locally. Defaults to false.
     exclude: A string or array of strings corresponding to rsync's --exclude option. Defaults to None.

@return bool: True if successful, False otherwise.
@par Example:
@code
{
    "command": "SSH_GetFile",
    "wait": 0,
    "data": {
"name": "workstation",
"remote_path": "./data/output.dat",
"local_path": "./results.txt"
    }
}
```

11.53.3.4 def plugins.ssh.ssh_plugin.SshPlugin.init_connection (*self*, *host*, *user* = None, *port* = None, *gateway* = None, *ssh_config_path* = None, *args* = None, *name* = "default")

```
Establishes an SSH connection with a target host.
This command must be run before other remote commands will work.
Command may be used multiple times with the same name to connect to different remote hosts in succession,
or be used with different names to maintain concurrent connections to multiple hosts.
   - **host**: hostname or IP to connect to, which may include the username and/or port.

@param name: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
@param user: User name for the connection. Do not use if you specified the user in `host`. (Optional)
@param port: Port number for the connection. Do not use if you specified the port in `host`. (Optional)
@param gateway: SSH gateway command string to proxy the connection to `host` (Optional)
@param ssh_config_path: Path to an ssh config file which may contain host definitions or additional parameters.
                If not specfied, `~/.ssh/config` will be assumed. (Optional)
@param args: Additional SSH connection options, as needed. See [Paramiko API docs] (Optional)
    (http://docs.paramiko.org/en/latest/api/client.html#paramiko.client.SSHClient.connect) for relevant values.

@return bool: True if successful, False otherwise.
```

11.53.3.5 def plugins.ssh.ssh_plugin.SshPlugin.initialize (*self*)

```
   Initialize implementation for the SSH plugin.

   @note The initialize function is called by the CTF plugin manager *after* all plugins have been loaded.

   @note This function may interact with other plugins, since all plugins have been loaded at this stage.

   @return bool: True if successful, False otherwise.
```

11.53.3.6 def plugins.ssh.ssh_plugin.SshPlugin.put_file (*self*, *local_path*, *remote_path*, *args* = None, *name* = "default")

```
Copies a path (file or directory) from the local filesystem to the remote host via rsync.
Relative or absolute paths are allowed, but do not use ~. Strings are passed directly to rsync,
so the same rules apply regarding paths, patterns, etc.

@param name: A name already registered with SSH_RegisterTarget to identify the connection. (Optional)
@param local_path: The path to the local file or directory to be copied.
@param remote_path: The path to where the file or directory is to be copied.
            For remote hosts use the SSH syntax user@host:path.
@param args: An object that describes optional parameters for the transfer.
     delete: A boolean corresponding to rsync's --delete option.
     If true, rsync will remove remote files that no longer exist locally. Defaults to false.
     exclude: A string or array of strings corresponding to rsync's --exclude option. Defaults to None.

@return bool: True if successful, False otherwise.
@par Example:
@code
{
    "command": "SSH_PutFile",
    "wait": 0,
    "data": {
"name": "workstation",
"local_path": "./cfs",
"remote_path": "/tmp/workspace/cfs",
"args": {
    "delete": true,
    "exclude": "*.git"
}
    }
}
```

11.53.3.7 def plugins.ssh.ssh_plugin.SshPlugin.register_target (*self*, *name* = "")

```
Declares a target host by name. This command must be run before any other commands given the same name.
```

```
Command may be used multiple times to declare any number of targets.
If not used,the plugin will assume that all commands are intended for the same target as defined in SSH_InitSSH.

@param name: An arbitrary, unique name to identify the target in subsequent commands.
Does not need be the actual hostname of the target. Name is optional in all other commands,
but if not provided all such commands will share a single connection.

@return bool: True if successful, False otherwise.

@par Example
@code
{
    "command": "SSH_RegisterTarget",
    "wait": 1,
    "data": {
 "name": "workstation"
    }
}
```

11.53.3.8 def plugins.ssh.ssh_plugin.SshPlugin.run_command (*self*, *command*, *cwd* = "", *prefix* = ":", *name* = "default")

```
Executes a command on the remote host. ExecutionInitSSH must be called first to establish an SSH connection.

@param name: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
@param command: The shell command to be executed. Can contain multiple commands separated with `;`
@return bool: True if successful, False otherwise.
@par Example:
@code
{
    "command": "SSH_RunLocalCommand",
    "wait": 1,
    "data": {
"name": "workstation",
"host": "cd lander_fsw_ctf/;rm -rf build; make; make install;"
    }
}
```

11.53.3.9 def plugins.ssh.ssh_plugin.SshPlugin.run_command_local (*self*, *command*, *name* = "default")

```
Executes a command on the local host (the machine running CTF), regardless of the target.
This is different from calling SSH_RunRemoteCommand targeting localhost,
as it is invoked directly by the current process rather than passed via SSH.

@param name: A name already registered with SSH_RegisterTarget to identify the connection. (Optional)
@param command: The shell command to be executed. Can contain multiple commands separated with ;
@return bool: True if successful, False otherwise.
@par Example:
@code
{
    "command": "SSH_RunLocalCommand",
    "wait": 1,
    "data": {
"name": "workstation",
"host": "cd lander_fsw_ctf/;rm -rf build; make; make install;"
    }
}
```

11.53.3.10 def plugins.ssh.ssh_plugin.SshPlugin.shutdown (*self*)

```
Shutdown implementation for the SSH plugin.
@note The shutdown function is called by the CTF plugin manager upon completion of a test run.
@note The shutdown function can be exposed to test scripts by adding it to the command map.
```

11.53.3.11 def plugins.ssh.ssh_plugin.SshPlugin.upload_ftp (*self*, *host*, *local_path*, *remote_path*, *name* = "default")

```
Uploads a path (file or directory) from the local filesystem to the FTP server.

@param name: A name already registered with `SSH_RegisterTarget` to identify the connection. (Optional)
@param host: The hostname or address of the FTP server.
@param remote_path: The path on the FTP server to where the file or directory is to be uploaded.
@param local_path: The local path to the source file or directory.

@return bool: True if successful, False otherwise.

@par Example:
@code
{
    "command": "SSH_PutFTP",
    "wait": 0,
    "data": {
"name": "workstation",
 "host": "ftphost",
 "remote_path": "./data/output.dat",
 "local_path": "./results.txt"
    }
}
```

11.54 lib.status.StatusDefs Class Reference

Static Public Attributes

- string **waiting** = 'waiting'
- string **active** = 'active'
- string **stopped** = 'stopped'
- string **passed** = 'passed'
- string **failed** = 'failed'
- string **error** = 'error'
- string **timeout** = 'timeout'
- string **aborted** = 'aborted'
- string **disabled** = 'disabled'

11.54.1 Detailed Description

```
This class defines enumerations for the status definitions used by CTF to send instruction status.
```

11.55 lib.status_manager.StatusManager Class Reference

Public Member Functions

- def __init__
- def start
- def set_scripts
- def update_suite_status
- def finalize_suite_status
- def update_script_status
- def update_test_status
- def update_command_status
- def end_command

- def end_test
- def end_script
- def sanitize_status
- def send_update

Static Public Member Functions

- def blank_status_msg
- def sanitize_param
- def sanitize_data

Data Fields

- **status**
- **script_index**
- **test_index**
- **command_index**
- **ip_address**
- **port**
- **socket**
- **start_time**

11.55.1 Detailed Description

```
The StatusManager class established a status stream with the current test suite status. The status packets are se
over a UDP socket over the specified port. Clients listening on that port will receive periodic CTF status messag
during test execution

@param ip_address: IP of the external listener to connect to
@param: port: Port used by the external listener to receive status messages
```

11.55.2 Constructor & Destructor Documentation

11.55.2.1 def lib.status_manager.StatusManager.__init__ (*self*, *ip_address* = "127.0.0.1", *port* = None)

```
Constructor of StatusManager Class: initiate instance properties.
```

11.55.3 Member Function Documentation

11.55.3.1 def lib.status_manager.StatusManager.blank_status_msg (*scripts*) [static]

```
Get a blank status message that contains status objects for each script loaded by CTF
```

11.55.3.2 def lib.status_manager.StatusManager.end_command (*self*)

```
Increment the current active command index.
```

11.55.3.3 def lib.status_manager.StatusManager.end_script (*self*)

```
Increment the current active script. Reset the test and command indices to 0.
```

11.55.3.4 def lib.status_manager.StatusManager.end_test (*self*)

Increment the current active test case index. Reset the command index to 0.

11.55.3.5 def lib.status_manager.StatusManager.finalize_suite_status (*self*)

Set the test suit status (pass/fail) based on the status of all scripts within the suite.

11.55.3.6 def lib.status_manager.StatusManager.sanitize_data (*data*) [static]

Sanitize test instruction data by attempting to decode every field if needed

11.55.3.7 def lib.status_manager.StatusManager.sanitize_param (*param*) [static]

Sanitize a test instruction parameter by attempting to decode it if needed

11.55.3.8 def lib.status_manager.StatusManager.sanitize_status (*self*)

Sanitize test script data by attempting to decode every field at the test script level if needed

11.55.3.9 def lib.status_manager.StatusManager.send_update (*self*)

Send the latest status packet over the UDP socket.

@note - If the UDP socket encounters an error for any reason, the port will be set to None and CTF will not send updates to the Editor any more. The socket failure is most likely to be a computer issue, not CTF issue.

11.55.3.10 def lib.status_manager.StatusManager.set_scripts (*self*, *scripts*)

Set the script status entry for each script with default values

11.55.3.11 def lib.status_manager.StatusManager.start (*self*)

Set the start time of test suite execution in the status message.

11.55.3.12 def lib.status_manager.StatusManager.update_command_status (*self*, *status*, *details*, *index* = None)

Update the status of a single command within a test script.

11.55.3.13 def lib.status_manager.StatusManager.update_script_status (*self*, *status*, *details* = "")

Update the status of a single script within the test suite.

11.55.3.14 def lib.status_manager.StatusManager.update_suite_status (*self*, *status*, *details*)

Given an updated status (and details), update the suite status with the latest state.

11.55.3.15 def lib.status_manager.StatusManager.update_test_status (*self*, *status*, *details* = "")

Update the status of a single script within the test suite.

11.56 plugins.cfs.pycfs.cfs_interface.TelemetryVerification Class Reference

Public Member Functions

- def __init__

Data Fields

- **verification_id**
- **condition**
- **passed**
- **pass_count**
- **fail_count**

11.56.1 Detailed Description

```
Telemetry Verification class
```

11.56.2 Constructor & Destructor Documentation

11.56.2.1 def plugins.cfs.pycfs.cfs_interface.TelemetryVerification.__init__ (*self*, *v_id*, *condition*)

```
Constructor for TelemetryVerification class.  Assign attribute default values.
```

11.57 lib.test.Test Class Reference

Public Member Functions

- def __init__
- def execute_instruction
- def execute_verification
- def process_verification_delay
- def run_commands
- def process_conditional_branch_label
- def process_control_flow_label
- def run_test

Static Public Member Functions

- def process_command_delay

Data Fields

- **test_info**
- **instructions**
- **test_result**
- **test_aborted**
- **test_run**
- **num_skipped**

- **num_ran**
- **test_start_time**
- **ctf_verification_timeout**
- **ctf_verification_poll_period**
- **end_test_on_fail**
- **ignored_instructions**
- **verif_list**
- **verify_required_commands**
- **continuous_verification_commands**
- **end_test_on_fail_commands**
- **status_manager**
- **current_instruction_index**

Static Private Member Functions

- def **__check_label_def**

11.57.1 Detailed Description

```
The TestCase class represents a CTF Test Case.
@note - A test script may have multiple test cases.
```

11.57.2 Constructor & Destructor Documentation

11.57.2.1 def lib.test.Test.__init__ (*self*)

```
Constructor of Test Class: Initiate test properties
```

11.57.3 Member Function Documentation

11.57.3.1 def lib.test.Test.execute_instruction (*self, test_instruction, command_index*)

```
Execute a CTF Test Instruction
```

11.57.3.2 def lib.test.Test.execute_verification (*self, command, command_index, timeout, new_verification =* `False`)

```
Execute a CTF Verification Instruction.
@note - Verification instructions will be executed at the specified poll period until the verification passes
or a timeout is reached
```

11.57.3.3 def lib.test.Test.process_command_delay (*delay*) [static]

```
Utilize the current CTF time manager to wait a specific amount of time before executing a CTF Test Instruction
```

11.57.3.4 def lib.test.Test.process_conditional_branch_label (*self*)

```
Process conditional branch labels defined in test instructions 'IfCondition', 'ElseCondition', 'EndCondition'
```

11.57.3.5 def lib.test.Test.process_control_flow_label (*self*)

```
Process control flow labels defined in test instructions 'BeginLoop' and 'EndLoop'
```

11.57.3.6 def lib.test.Test.process_verification_delay (*self*)

```
Utilize the current CTF time manager to wait for the duration of the polling period before executing a CTF
Verification Test Instruction
```

11.57.3.7 def lib.test.Test.run_commands (*self*)

```
Run all CTF Instructions in the current test case
```

11.57.3.8 def lib.test.Test.run_test (*self*, *status_manager*)

```
Run all CTF Instructions within a test case
```

11.58 lib.logger.TestFormatter Class Reference

Public Member Functions

- def **formatTime**

11.58.1 Detailed Description

```
TestFormatter: Customizes the logging formatter to override formatTime
```

11.59 lib.test_script.TestScript Class Reference

Public Member Functions

- def __init__
- def set_header_info
- def set_options
- def set_watch_lists
- def set_tests
- def run_script
- def log_test_header
- def generate_test_results

Data Fields

- **test_number**
- **test_name**
- **requirements**
- **test_description**
- **options**
- **telem_watch_list**
- **cmd_watch_list**
- **test_owner**
- **test_setup**
- **verify_timeout**
- **tests**
- **input_file_path**

- **input_file**
- **params**
- **status**
- **start_time**
- **exec_time**
- **num_tests**
- **num_passed**
- **num_failed**
- **num_error**

11.59.1 Detailed Description

```
The TestScript class represents a CTF test script, storing script data and status.
```

11.59.2 Constructor & Destructor Documentation

11.59.2.1 def lib.test_script.TestScript.__init__ (*self*)

```
Constructor of TestScript Class: Initiate instance properties
```

11.59.3 Member Function Documentation

11.59.3.1 def lib.test_script.TestScript.generate_test_results (*self*)

```
Generate and Log the test results after test execution
```

11.59.3.2 def lib.test_script.TestScript.log_test_header (*self*)

```
Log the test header (metadata) before beginning test execution
```

11.59.3.3 def lib.test_script.TestScript.run_script (*self, status_manager*)

```
Execute a complete test script, updating the status_manager as needed.
```

11.59.3.4 def lib.test_script.TestScript.set_header_info (*self, test_number, test_name, requirements, test_description, test_owner, test_setup, verify_timeout*)

```
Set the TestScript's header information from the input test script file.

@param test_number: Test number
@param test_name: Test name
@param requirements: Requirements validated by this test
@param test_description: Test Description
@param test_owner: Test Owner
@param test_setup: Test Setup
@param verify_timeout: Test Specific Verification Timeout (Overrides 'ctf_verification_timeout' in INI File)
```

11.59.3.5 def lib.test_script.TestScript.set_options (*self, options*)

```
Set the TestScript's options from the input test script file.
@param options: Test Script Options (Dict)
```

11.59.3.6 def lib.test_script.TestScript.set_tests (*self*, *tests*)

```
Set the list of test cases within this test script
```

11.59.3.7 def lib.test_script.TestScript.set_watch_lists (*self*, *telem_watch_list*, *cmd_watch_list*)

```
Set the TestScript's telemetry and command watch lists.
@note Telemetry and Command watch list are currently not used by CTF.
@param telem_watch_list: Test Script Telemetry Watch List
@param cmd_watch_list: Test Script Command Watch List
```

11.60 lib.time_interface.TimeInterface Class Reference

Public Member Functions

- def __init__
- def wait

Static Public Member Functions

- def wait_seconds
- def pre_command
- def post_command

Data Fields

- exec_time
  Execution time since the time manager was initialized.
- last_command_completion_time
  Execution time when the last instruction was completed.
- time_since_last_command
  How much time has passed since the last instruction was completed.

11.60.1 Detailed Description

```
Virtual class definition for custom plugins to implement their own time managers.

@note A custom plugin must set the global time manager used by CTF using Global.set_time_manager(time_manager)
```

11.60.2 Constructor & Destructor Documentation

11.60.2.1 def lib.time_interface.TimeInterface.__init__ (*self*)

```
Constructor of TimeInterface Class: Initiate instance properties
```

11.60.3 Member Function Documentation

11.60.3.1 def lib.time_interface.TimeInterface.post_command ( ) [static]

```
Optional implementation of logic to be executed *after* a CTF instruction is invoked.

@note - This is useful when pausing/resuming of frames on an external time source is needed.
```

11.60.3.2 def lib.time_interface.TimeInterface.pre_command ( ) [static]

```
Optional implementation of logic to be executed *before* a CTF instruction is invoked.

@note - This is useful when pausing/resuming of frames on an external time source is needed.
```

11.60.3.3 def lib.time_interface.TimeInterface.wait (*self*, *seconds*)

```
Virtual method to wait an amount of time.

@note - May include special logic to interface with external time sources
```

11.60.3.4 def lib.time_interface.TimeInterface.wait_seconds (*seconds*) [static]

```
Helper utility to wait in seconds (OS Time)
```

11.60.4 Field Documentation

11.60.4.1 lib.time_interface.TimeInterface.time_since_last_command

How much time has passed since the last instruction was completed.

11.61 plugins.cfs.pycfs.tlm_listener.TlmListener Class Reference

Public Member Functions

- def __init__
- def cleanup
- def create_socket
- def get_port
- def read_socket

Data Fields

- **ipaddr**
- **port**
- **socket**

11.61.1 Detailed Description

```
Simple telemetry listener class that connects to a given ip/port via UDP and manages that connection.
Can call read_socket() to receive the next packet in telemetry stream.
```

11.61.2 Constructor & Destructor Documentation

11.61.2.1 def plugins.cfs.pycfs.tlm_listener.TlmListener.__init__ (*self*, *ipaddr*, *port*)

```
Constructor of TlmListener class.
@param ipaddr: IP address of cFS system.
@param port:   port of cFS system.
@return None
```

11.61.3 Member Function Documentation

11.61.3.1 def plugins.cfs.pycfs.tlm_listener.TlmListener.cleanup (*self*)

```
Close socket connection.

@return None
```

11.61.3.2 def plugins.cfs.pycfs.tlm_listener.TlmListener.create_socket (*self*)

```
Create a UDP socket connection to a cFS system.
@return socket
```

11.61.3.3 def plugins.cfs.pycfs.tlm_listener.TlmListener.get_port (*self*)

```
Return the UDP port to cFS system
@return UDP port
```

11.61.3.4 def plugins.cfs.pycfs.tlm_listener.TlmListener.read_socket (*self*)

```
Receive the UDP packet in the telemetry stream.

@return the number of bytes read from telemetry stream
```

11.62 plugins.cfs.pycfs.output_app_interface.ToApi Class Reference

Public Member Functions

- def __init__
- def disable_output
- def enable_output

Data Fields

- **command_args**
- **cmd_cc**
- **mid**
- **name**

11.62.1 Detailed Description

```
Construct the ToApi class

For CFS, TO is used to extract command and telemetry CCSDS packets from the software bus, and is sent over UDP
to the CFS test framework.
```

11.62.2 Constructor & Destructor Documentation

11.62.2.1 def plugins.cfs.pycfs.output_app_interface.ToApi.__init__ (*self*, *local_ip* = " ", *local_port* = 0, *command_interface* = None, *ccsds_ver* = 0, *mid_map* = None, *name* = None)

```
Constructor of the ToApi class.
```

```
@param local_ip: The IP address we want packets to be forwarded to. Default: 127.0.0.1
@param local_port: The port we want packets to be forwarded to. Default: 40096
@param command_interface: An instance of the CommandInterface class (used to send commands to UDP)
@param ccsds_ver: CCSDS header version (1 or 2)
```

11.62.3 Member Function Documentation

11.62.3.1 def plugins.cfs.pycfs.output_app_interface.ToApi.disable_output (*self*)

```
disable_output cFS instruction is not implemented in ToApi class, always return True.
@return bool: always return True
```

11.62.3.2 def plugins.cfs.pycfs.output_app_interface.ToApi.enable_output (*self*)

```
Implement enable_output method for ToApi class.
Build "SendCfsCommand" instruction with command code "TO_ENABLE_OUTPUT",
search for a plugin to send out the instruction.
@return bool: True if a plugin send out instruction successfully; otherwise False
```

11.63 plugins.userio_plugin.userio_plugin.UserIOPlugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def shutdown

Static Public Member Functions

- def waituserinput_command

Data Fields

- name

  Plugin Name.
- description

  Plugin Description.
- command_map

  Plugin Command Map.
- end_test_on_fail_commands

  List of end_test_on_fail_commands commands.

11.63.1 Detailed Description

```
The UserIO Plugin Class Definition

@note The UserIO Plugin define a command to allow user to pause the testing. User must confirm to continue testin
       for safety critical tasks.

@note The CTF will wait until a user instructs to continue or abort the testing. If aborting the testing,
      the tests after the instruction will not be executed.
```

```
@note The plugin adds a new command in end_test_on_fail_commands for test.py to check the user input.

@note The custom plugin class *must* inherit from the Plugin base-class.

@note A custom CTF plugin can be created to add new CTF instructions that can then be utilized within a JSON test
    script.

@note All plugin functions mapped to a test instruction *must* return true/false to indicate pass/fail of
    that instruction.
```

11.63.2 Constructor & Destructor Documentation

11.63.2.1 def plugins.userio_plugin.userio_plugin.UserIOPlugin.__init__ (*self*)

```
Constructor implementation for example plugin.

@note The __init__ function is called once a plugin is loaded.

@note The __init__ function should not reference/interact with any other plugin since the other plugin may not
    be loaded at this stage.

@note The constructor of a plugin must define the following fields:
    - name
    - description
    - command map: dictionary mapping CTF instructions to a tuple defining the
          python function to use for that instruction, and a list of argument types
    - [optional] verify_required_commands: List of instructions that require verification (i.e polling
    until verification passes or timeout.
    - other class variables that can store state, etc...
```

11.63.3 Member Function Documentation

11.63.3.1 def plugins.userio_plugin.userio_plugin.UserIOPlugin.initialize (*self*)

```
Initialize implementation for the UserIO plugin.

@note The initialize function is called by the CTF plugin manager *after* all plugins have been loaded.

@note This function may interact with other plugins, since all plugins have been loaded at this stage.

@return bool: True if successful, False otherwise.
```

11.63.3.2 def plugins.userio_plugin.userio_plugin.UserIOPlugin.shutdown (*self*)

```
Shutdown implementation for the userio plugin.
@note The shutdown function is called by the CTF plugin manager upon completion of a test run.
@note The shutdown function can be exposed to test scripts by adding it to the command map.
```

11.63.3.3 def plugins.userio_plugin.userio_plugin.UserIOPlugin.waituserinput_command (*prompt* = "") [static]

```
Wait for user input:  if there is no user input, wait forever;
if user input is 'Y' or 'y', continue the test;
if user input is anything else, abort the test

@param prompt: any value (example: "user input")

@return bool: True if successful, False otherwise.
```

11.64 plugins.validation_plugin.validation_plugin.ValidationPlugin Class Reference

Public Member Functions

- def __init__
- def **save_file_as_text**
- def interpret_binary_data
- def **interpret_event_log**
- def shutdown

Static Public Member Functions

- def initialize
- def **delete_file**
- def **copy_file**
- def convert_timestamp
- def **parse_txt_file**

Data Fields

- name

  Plugin Name.
- description

  Plugin Description.
- command_map

  Plugin Command Map.

11.64.1 Detailed Description

```
The Validation Plugin Class Definition

@note The plugin uses const values / macros to interpret log files,
       some of these values comes from registered target's controller (which reads ccdd json files).
       So InterpretCfsFile instruction could NOT be used before RegisterCfs/StartCfs instructions.
```

11.64.2 Constructor & Destructor Documentation

11.64.2.1 def plugins.validation_plugin.validation_plugin.ValidationPlugin.__init__ (*self*)

```
Constructor implementation for validation plugin.
```

11.64.3 Member Function Documentation

11.64.3.1 def plugins.validation_plugin.validation_plugin.ValidationPlugin.convert_timestamp (*subsecs, str*) [static]

```
Helper function: convert subsecs to microseconds
```

11.64.3.2 def plugins.validation_plugin.validation_plugin.ValidationPlugin.initialize (*bool*) [static]

```
Initialize implementation for the validation plugin.

@note The initialize function is called by the CTF plugin manager *after* all plugins have been loaded.

@return bool: True
```

11.64.3.3 def plugins.validation_plugin.validation_plugin.ValidationPlugin.interpret_binary_data (*self*, *event_data*, *entry_id*, *offset*, *endianess_of_target*, *os_max_api_name*, *str*)

```
Helper function: interpret each line of the binary cFE Event Log data to a human readable message
```

11.64.3.4 def plugins.validation_plugin.validation_plugin.ValidationPlugin.shutdown (*self*)

```
Shutdown implementation for the validation plugin.
@note The shutdown function is called by the CTF plugin manager upon completion of a test run.
@note The shutdown function can be exposed to test scripts by adding it to the command map.
```

11.65 plugins.variable_plugin.variable_plugin.VariablePlugin Class Reference

Public Member Functions

- def __init__
- def initialize
- def shutdown

Static Public Member Functions

- def **set_user_defined_variable**
- def **set_user_variable_from_tlm**
- def **set_user_variable_from_tlm_header**
- def **set_label**
- def **get_user_defined_variable**
- def **check_user_defined_variable**

Data Fields

- name

    Plugin Name.
- description

    Plugin Description.
- command_map

    Plugin Command Map.

11.65.1 Detailed Description

```
The Variable Plugin Class Definition

@note The Variable Plugin allows users to set / read / test variables defined in json test scripts.

@note All plugin functions mapped to a test instruction *must* return true/false to indicate pass/fail of
    that instruction.
```

11.65.2 Constructor & Destructor Documentation

11.65.2.1 def plugins.variable_plugin.variable_plugin.VariablePlugin.__init__ (*self*)

```
Constructor of variable plugin.

@note The __init__ function is called once a plugin is loaded.

@note The __init__ function should not reference/interact with any other plugin since the other plugin may not
    be loaded at this stage.

@note The constructor of a plugin must define the following fields:
    - name
    - description
    - command map: dictionary mapping CTF instructions to a tuple defining the
          python function to use for that instruction, and a list of argument types
    - [optional] verify_required_commands: List of instructions that require verification (i.e polling
    until verification passes or timeout.
    - other class variables that can store state, etc...
```

11.65.3 Member Function Documentation

11.65.3.1 def plugins.variable_plugin.variable_plugin.VariablePlugin.initialize (*self*)

```
Initialize implementation for the variable plugin.

@note The initialize function is called by the CTF plugin manager *after* all plugins have been loaded.

@note This function may interact with other plugins, since all plugins have been loaded at this stage.

@return bool: True if successful, False otherwise.
```

11.65.3.2 def plugins.variable_plugin.variable_plugin.VariablePlugin.shutdown (*self*)

```
Shutdown implementation for the variable plugin.
@note The shutdown function is called by the CTF plugin manager upon completion of a test run.
@note The shutdown function can be exposed to test scripts by adding it to the command map.
```

Index