

**tmtccmd**

***Release 8.0.1***

**Robin Mueller**

**May 16, 2024**



# CONTENTS

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Introduction . . . . .	3
1.2	Getting Started . . . . .	4
1.3	Communication Abstraction . . . . .	7
1.4	CCSDS File Delivery Protocol (CFDP) . . . . .	10
1.5	API . . . . .	10
<b>2</b>	<b>Indices and tables</b>	<b>69</b>
	<b>Python Module Index</b>	<b>71</b>
	<b>Index</b>	<b>73</b>



This is a small Python framework targeted towards the testing of remote systems like satellites and rovers. It simplifies sending and receiving TMTCs (Telemetry and Telecommands) and testing via different communication interfaces. This tool can be used either as a command line tool or as a GUI tool which requires a PyQt5 installation. This package also has dedicated support to send and receive ECSS PUS packets or other generic CCSDS packets.

This framework started as a small application for the [SOURCE](#) project to test the on-board software but has evolved to be more generic and easily adaptable to new projects.

Other pages (online)

- [project page on GitHub](#)
- This page, when viewed online is at <https://tmtccmd.readthedocs.io/en/latest/>

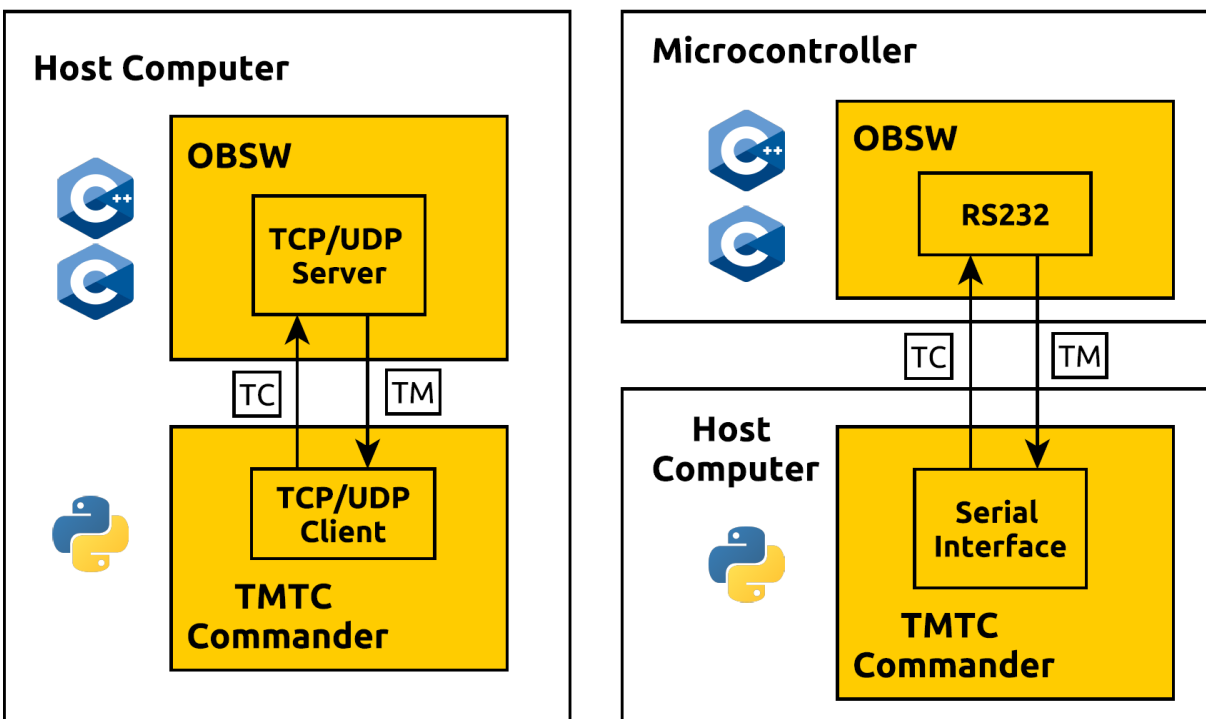


## CONTENTS

## 1.1 Introduction

### 1.1.1 Overview

The goal of this framework is to make it as easy to send telecommands (TCs) to the On-Board Software (OBSW) running on an external On-Board Computer (OBC) and to analyse the telemetry (TMs) coming back. The following graph shows two possible ways to use the TMTC commander



The first way assumes that the OBSW can be run on a host computer and starts a TCP/IP server internally. The TMTC commander can then be used to send telecommands via the TCP/IP interface. The second way assumes that the OBSW is run on an external microcontroller. Here, the serial interface is used to send telecommands. Other ways like sending TMTCs via Ethernet to a microcontroller running a TCP/IP server are possible as well.

## 1.1.2 Features

- Generic communication interface abstraction in form of the `tmtccmd.com.ComInterface`. This abstraction could also be used without the other components of the library if the goal is to separate the packet logic from the communication interface. The *Communication Abstraction* chapter contains a more information and examples.
- Special support for *Packet Utilisation Standard (PUS)* packets and *CCSDS Space Packets*. This library uses the `spacepackets` library for most packet implementations.
- Support for both CLI and GUI usage.
- Flexibility in the way to specify telecommands to send and how to handle incoming telemetry. This is done by requiring the user to specify callbacks for both TC specification and TM handling.
- One-Queue Mode for simple command sequences and Multi-Queue for more complex command sequences.
- Listener mode to only listen to incoming telemetry.
- Some components are tailored towards usage with the *Flight Software Framework (FSFW)* and the `sat-rs` library.

This framework also has a communication interface abstraction which allows to exchange TMTC through different channels. The framework currently supports (among others) the following communication interfaces:

1. TCP/IP with the `tmtccmd.com.udp.UdpClient` and `tmtccmd.com.tcp.TcpSpacepacketsClient`.
2. Serial Communication with COBS encoded packets by using the `tmtccmd.com.serial_cobs.SerialCobsComIF`.

It is also possible to supply custom interfaces.

## 1.2 Getting Started

### 1.2.1 Example Project

The *example application* is the best way to learn how this framework works and to get started. It shows how to set up handler classes for TC and TM handling and then ties together all components. You can also run this application in GUI mode by passing the `-g` GUI flag to the example application.

Some explanation of classes and modules inside the example are given here.

### 1.2.2 The Configuration Hook Class

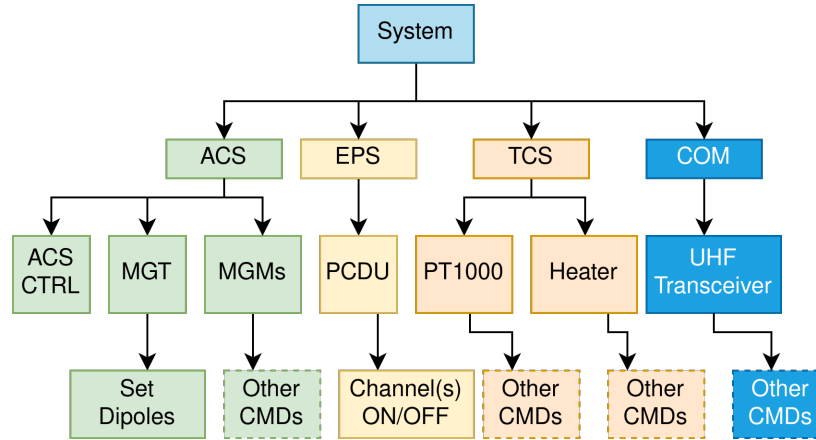
The class `ExampleHookClass` is the example configuration class implementing the `tmtccmd.config.hook.HookBase`. There are two functions which must be implemented by a user:

1. The `tmtccmd.config.hook.HookBase.get_communication_interface()` method is used to return a communication interface given a string identifier. You can read more about the communication abstraction in the *Communication Abstraction* chapter.
2. The `tmtccmd.config.hook.HookBase.get_command_definitions()` returns a tree of command definition which can be used by both users and developers to specify the available commands and procedures.



## TC Command Definition Specification

The command tree mechanism provides a flexible mechanism to also model the command definitions for more complex systems. These systems are oftentimes structured into dedicated modules. For example, the command tree for a satellite system might look like this:



For a system like this, it makes a lot of sense to also model the command definitions similarly to the system. An example tree modelling the system shown in the example above would look like this when printing one built using the `tmtccmd.config.tmtc.CmdTreeNode`:

```

/
├── ping
│   └── [ Root Node ]
│       └── [ Send PUS ping command ]
├── test
│   └── [ Test Node ]
│       └── [ Send PUS event test command ]
├── system
│   └── [ System Commands ]
├── acs
│   └── [ ACS Subsystem ]
│       ├── acs_ctrl
│       │   └── [ ACS Controller ]
│       ├── mgt
│       │   └── [ Magnetorquer ]
│       │       └── set_dipoles
│       │           └── [ Set MGT Dipoles ]
│       ├── mgm0
│       │   └── [ Magnetometer 0 ]
│       │       └── other cmds
│       │           └── [ Other MGM commands ]
│       ├── mgm1
│       │   └── [ Magnetometer 1 ]
│       │       └── other cmds
│       │           └── [ Other MGM commands ]
├── tcs
│   └── [ TCS Subsystem ]
│       ├── tcs_ctrl
│       │   └── [ TCS Controller ]
│       ├── pt1000
│       │   └── [ Temperature Sensor ]
│       └── heater
│           └── [ Heater ]
├── com
│   └── [ COM Subsystem ]
│       └── uhf_transceiver
│           └── [ UHF Transceiver ]
├── eps
│   └── [ EPS Subsystem ]
│       └── pcd
│           └── [ PCDU ]
│               ├── channel_0_on
│               │   └── [ Channel 0 on ]
│               ├── channel_0_off
│               │   └── [ Channel 0 off ]
│               ├── channel_1_on
│               │   └── [ Channel 1 on ]
│               └── channel_1_off
│                   └── [ Channel 1 off ]

```

and the code to create this tree would look like this:

```

def get_command_definitions(self) -> CmdTreeNode:
    root_node = CmdTreeNode.root_node()

```

(continues on next page)

(continued from previous page)

```

root_node.add_child(CmdTreeNode("ping", "Send PUS ping command"))
root_node.add_child(CmdTreeNode("test", "Test Node"))
root_node.children["test"].add_child(
    CmdTreeNode("event", "Send PUS event test command")
)
root_node.add_child(CmdTreeNode("system", "System Commands"))
root_node.add_child(CmdTreeNode("acs", "ACS Subsystem"))
root_node["acs"].add_child(CmdTreeNode("acs_ctrl", "ACS Controller"))
root_node["acs"].add_child(CmdTreeNode("mgt", "Magnetorquer"))
root_node["acs"]["mgt"].add_child(CmdTreeNode("set_dipoles", "Set MGT Dipoles"))
root_node["acs"].add_child(CmdTreeNode("mgm0", "Magnetometer 0"))
root_node["acs"].add_child(CmdTreeNode("mgm1", "Magnetometer 1"))
mgm_node = CmdTreeNode("other cmds", "Other MGM commands")
root_node["acs"]["mgm0"].add_child(mgm_node)
root_node["acs"]["mgm1"].add_child(mgm_node)
root_node.add_child(CmdTreeNode("tcs", "TCS Subsystem"))
root_node["tcs"].add_child(CmdTreeNode("tcs_ctrl", "TCS Controller"))
root_node["tcs"].add_child(CmdTreeNode("pt1000", "Temperature Sensor"))
root_node["tcs"].add_child(CmdTreeNode("heater", "Heater"))
root_node.add_child(CmdTreeNode("com", "COM Subsystem"))
root_node["com"].add_child(CmdTreeNode("uhf_transceiver", "UHF Transceiver"))
root_node.add_child(CmdTreeNode("eps", "EPS Subsystem"))
root_node["eps"].add_child(CmdTreeNode("pcdu", "PCDU"))
root_node["eps"]["pcdu"].add_child(CmdTreeNode("channel_0_on", "Channel 0 on"))
root_node["eps"]["pcdu"].add_child(
    CmdTreeNode("channel_0_off", "Channel 0 off")
)
root_node["eps"]["pcdu"].add_child(CmdTreeNode("channel_1_on", "Channel 1 on"))
root_node["eps"]["pcdu"].add_child(
    CmdTreeNode("channel_1_off", "Channel 1 off")
)
return root_node

```

You can now specify your commands as command paths, which will then serve as identifier for single command or command stacks and procedures. The command path will be passed on as the *cmd\_path* parameter of the `tmtccmd.tmtc.procedure.DefaultProcedureInfo` which is passed to the `tmtccmd.tmtc.handler.TcHandlerBase` implementation of the user.

It is also possible to pass the command path as a CLI argument. For example, you can use `./tmtcc.py -p /test` to send a ping command with the example application. Passing the command tree definition to the framework also allows it to provide a GUI command path selector for the GUI mode.

It is optionally possible to pass a command history to the framework by implementing the `tmtccmd.config.hook.HookBase.get_cmd_history()` function. An example implementation using the `prompt_toolkit.history.FileHistory` class would look like this:

```

def get_cmd_history(self) -> Optional[History]:
    """Optionally return a history class for the past command paths which will be used
    when prompting a command path from the user in CLI mode."""
    return FileHistory(".tmtc-cli-history.txt")

```

### 1.2.3 The TC handler

This object is responsible for the telecommand handling. Therefore this object implements the `tmtccmd.tmtc.handler.TcHandlerBase`.

In the example case, the handler object is responsible for returning telecommand queues based on input information. This task is done by the `tmtccmd.tmtc.handler.TcHandlerBase.feed_cb()` callback method.

The actual handling of telecommand queue entries is done in the `tmtccmd.tmtc.handler.TcHandlerBase.send_cb()` method implementation. One thing to note here is that a queue entry does not necessarily have to be a command to be sent. For example, the queue can also contain something like log requests or delay requests, or even complete custom requests. These requests can then be handled by the user.

### 1.2.4 The PUS TM handler

This object is responsible for the handling of PUS telemetry. In the example case, the handler object is responsible space packets with a certain application process identifier (APID). Therefore, this object implements the `tmtccmd.tmtc.common.SpecificApidHandlerBase`.

The `handle_tm` method implementation is the primary functions where incoming PUS packets are handled. This can something like prinouts or logging, either to a file or to a database.

### 1.2.5 Other example applications

The [EIVE](#) and [SOURCE](#) project implementation of the TMTC commander provide more complex implementations.

## 1.3 Communication Abstraction

This library contains a generic communication abstraction specifically targeted towards the exchange of binary data like CCSDS packets.

The core of this abstraction is the `tmtccmd.com.ComInterface` class. An implementation of this class can be passed to other library components and makes the commanding logic independent of the used interface. It is also possible to use this abstraction independently of the other library components and implement custom interfaces.

The use of a communication abstraction allows to use the same TMTC specification independently of the used communication interface. For example, it might be possible to build an on-board software both for a remote MCU and for a hosted system. The MCU might expect the same command data to be exchanged via a specific transport layer, while the hosted version might just use a UDP socket.

The following example shows how to use the `tmtccmd.com.udp.UdpClient` to send PUS packets (a subtype of CCSDS space packets) to a UDP server

```
import socket

from tmtccmd.com import ComInterface
from tmtccmd.com.udp import UdpClient, EthAddr
from spacepackets.ecss.tc import PusTelecommand

def send_my_telecommand(tc: PusTelecommand, com_if: ComInterface):
    com_if.send(tc.pack())

simulated_udp_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

(continues on next page)

(continued from previous page)

```
# Let the OS assign an appropriate port.
addr = ("127.0.0.1", 0)
simulated_udp_server.bind(addr)
addr = simulated_udp_server.getsockname()

udp_client = UdpClient("udp", send_address=EthAddr.from_tuple(addr))
udp_client.initialize()
udp_client.open()

ping_tc = PusTelecommand(service=17, subservice=1, apid=0x020)
send_my_telecommand(ping_tc, udp_client)

recvd_data, sender_addr = simulated_udp_server.recvfrom(4096)
recvd_tc = PusTelecommand.unpack(recvd_data)

print(f"Sent TC: {ping_tc}")
print(f"Received TC: {recvd_tc}")
udp_client.close()
```

Output:

```
Sent TC: PUS TC[17, 1] with Request ID 0x1820c000, APID 0x020, SSC 0
Received TC: PUS TC[17, 1] with Request ID 0x1820c000, APID 0x020, SSC 0
```

It should be noted that the `tmtccmd.com.ComInterface.receive` function should never block and returns a list of received packets. For many concrete implementations, this means that a separate receiver thread is required to poll for packets periodically and fill them into a packet list, which is then returned on the receive call.

The receiver thread may then also implement the logic required for some transport layers using blocking API. For example, the serial COBS interface will perform a blocking `serial.Serial.read()` to look for the start marker 0, and then read until the end marker 0 has been read.

Here is another example where a the same packet is sent via a serial interface. This example only runs on Unix systems because it simulates a serial port using `pty`.

```
import os
import pty
import time
from cobs import cobs

from tmtccmd.com import ComInterface
from tmtccmd.com.serial_cobs import SerialCfg, SerialCobsComIF

sim_ser_device, pty_slave = pty.openpty()
sim_serial_port = os.ttyname(pty_slave)
ser_cfg = SerialCfg(
    "serial_cobs", serial_port=sim_serial_port, baud_rate=9600, serial_timeout=1.0
)
cobs_com_if = SerialCobsComIF(ser_cfg)
cobs_com_if.initialize()
cobs_com_if.open()

test_data = bytes([0x01, 0x02, 0x03])
```

(continues on next page)

(continued from previous page)

```

# Will be used later and to determine how much to read.
encoded_data = cobs.encode(test_data)

# Data will be COBS encoded internally, with the 0 frame delimiter inserted at the start.
↪ and
# end
print(f"Sending raw data: 0x[{test_data.hex(sep=',')}]")
cobs_com_if.send(test_data)

# Other side receives COBS encoded packet
encoded_packet = os.read(sim_ser_device, len(encoded_data) + 2)
decoded_packet = cobs.decode(encoded_packet[1:-1])
print(f"Encoded packet received at simulated serial device: 0x[{encoded_packet.hex(sep=',')}
↪ ')']")
print(f"Decoded packet: 0x[{decoded_packet.hex(sep=',')}]")

# Now send COBS encoded data back
data_sent_back = bytes([0x01, 0x02, 0x03])
# 0 start marker
cobs_encoded_data = bytearray([0])
cobs_encoded_data.extend(encoded_data)
# 0 end marker
cobs_encoded_data.append(0)
os.write(sim_ser_device, cobs_encoded_data)
# Receiver thread might take some time
time.sleep(0.1)

packet_list = cobs_com_if.receive()
print(f"Data received from simulated serial device: 0x[{packet_list[0].hex(sep=',')}]")
cobs_com_if.close()

```

Output:

```

Sending raw data: 0x[01,02,03]
Encoded packet received at simulated serial device: 0x[00,04,01,02,03,00]
Decoded packet: 0x[01,02,03]
Data received from simulated serial device: 0x[01,02,03]

```

This interface could of course also exchange a higher level protocol like PUS packets, but this example was kept more simple to also show how a communication interface can also provide a transport layer.

## 1.4 CCSDS File Delivery Protocol (CFDP)

The majority of the CFDP components were moved to the dedicated `cfdp-py` library. It is recommended to read its [dedicated documentation](#).

## 1.5 API

### 1.5.1 Communication Package

Communication module. Provides generic abstraction for communication and commonly used concrete implementations.

#### Communication Package

Communication module. Provides generic abstraction for communication and commonly used concrete implementations.

**class** `tmtccmd.com.ComInterface`

Bases: `ABC`

Generic form of a communication interface to separate communication logic from the underlying interface.

**abstract** `close(args: Any = 0)`

Closes the ComIF and releases any held resources (for example a Communication Port).

#### Returns

**abstract** `data_available(timeout: float, parameters: Any = 0) → int`

Check whether TM packets are available.

#### Parameters

- **timeout** – Can be used to block on available data if supported by the specific communication interface.
- **parameters** – Can be an arbitrary parameter.

#### Raises

**`ReceptionDecodeError`** – If the underlying COM interface uses encoding and decoding when determining the number of available packets, this exception can be thrown on decoding errors.

#### Returns

0 if no data is available, number of packets otherwise.

**abstract** `property id: str`

**abstract** `initialize(args: Any = 0) → Any`

Perform initializations step which can not be done in constructor or which require returnvalues.

**abstract** `is_open() → bool`

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**abstract open**(args: *Any* = 0)

Opens the communication interface to allow communication.

**Returns**

**abstract receive**(parameters: *Any* = 0) → List[bytes]

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

**Parameters**

**parameters**

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

**Returns**

**abstract send**(data: *bytes*)

Send raw data.

**Raises**

**SendError** – Sending failed for some reason.

**exception** tmtccmd.com.ReceptionDecodeError(msg: *str*, custom\_exception: *Exception* | *None*)

Bases: *Exception*

Generic decode error which can also wrap the exception thrown by other libraries.

**exception** tmtccmd.com.SendError(msg: *str*, custom\_exception: *Exception* | *None*)

Bases: *Exception*

Generic send error which can also wrap the exception thrown by other libraries.

## TCP Client Module

TCP communication interface

**class** tmtccmd.com.tcp.TcpCommunicationType(value)

Bases: *Enum*

Parse for space packets in the TCP stream, using the space packet header.

**SPACE\_PACKETS** = 0

**class** tmtccmd.com.tcp.TcpSpacepacketsClient(com\_if\_id: *str*, space\_packet\_ids: *Sequence*[PacketId],  
inner\_thread\_delay: *float*, target\_address: *EthAddr*,  
max\_packets\_stored: *int* | *None* = *None*)

Bases: *ComInterface*

Communication interface for TCP communication. This particular interface expects raw space packets to be sent via TCP and uses a list of passed packet IDs to parse for them.

**close**(args: *any* | *None* = *None*) → *None*

Closes the ComIF and releases any held resources (for example a Communication Port).

**Returns**

**data\_available**(*timeout: float = 0, parameters: any = 0*) → int

Check whether TM packets are available.

**Parameters**

- **timeout** – Can be used to block on available data if supported by the specific communication interface.
- **parameters** – Can be an arbitrary parameter.

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding when determining the number of available packets, this exception can be thrown on decoding errors.

**Returns**

0 if no data is available, number of packets otherwise.

**property id:** str

**initialize**(*args: Any | None = None*)

Perform initializations step which can not be done in constructor or which require returnvalues.

**is\_open**() → bool

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**open**(*args: Any | None = None*)

Opens the communication interface to allow communication.

**Returns**

**receive**(*poll\_timeout: float = 0*) → List[bytes]

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

**Parameters**

**parameters**

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

**Returns**

**send**(*data: bytes*)

Send raw data.

**Raises**

**SendError** – Sending failed for some reason.

**class** tmtccmd.com.tcpip\_utils.**EthAddr**(*ip\_addr: 'str', port: 'int'*)

Bases: object

**classmethod** **from\_tuple**(*addr: Tuple[str, int]*) → EthAddr

**ip\_addr:** str

**port:** int

**property** **to\_tuple:** Tuple[str, int]



```
class tmtccmd.com.tcpip_utils.TcpIpConfigIds(value)
```

Bases: [Enum](#)

An enumeration.

**RECV\_ADDRESS** = 2

**RECV\_MAX\_SIZE** = 3

**SEND\_ADDRESS** = 1

**SPACE\_PACKET\_ID** = 4

**auto** = <class 'enum.auto'>

```
class tmtccmd.com.tcpip_utils.TcpIpType(value)
```

Bases: [Enum](#)

An enumeration.

**TCP** = 1

**UDP** = 2

**UDP\_RECV** = 3

```
tmtccmd.com.tcpip_utils.determine_recv_buffer_len(json_cfg_path: str, tcpip_type: TcpIpType)
```

```
tmtccmd.com.tcpip_utils.determine_tcp_send_address(json_cfg_path: str) → EthAddr
```

```
tmtccmd.com.tcpip_utils.determine_tcpip_address(tcpip_type: TcpIpType, json_cfg_path: str) →  
EthAddr
```

```
tmtccmd.com.tcpip_utils.determine_udp_recv_address(json_cfg_path: str) → EthAddr
```

```
tmtccmd.com.tcpip_utils.determine_udp_send_address(json_cfg_path: str) → EthAddr
```

```
tmtccmd.com.tcpip_utils.prompt_ip_address(type_str: str) → EthAddr
```

Prompt a valid IP address from the user

```
tmtccmd.com.tcpip_utils.prompt_recv_buffer_len(tcpip_type: TcpIpType) → int
```

## UDP Client Module

UDP Communication Interface

```
class tmtccmd.com.udp.UdpClient(com_if_id: str, send_address: EthAddr, recv_addr: EthAddr | None =  
None)
```

Bases: [ComInterface](#)

Communication interface for UDP communication

```
close(args: any | None = None) → None
```

Closes the ComIF and releases any held resources (for example a Communication Port).

**Returns**

**data\_available**(*timeout: float = 0, parameters: any = 0*) → bool

Check whether TM packets are available.

**Parameters**

- **timeout** – Can be used to block on available data if supported by the specific communication interface.
- **parameters** – Can be an arbitrary parameter.

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding when determining the number of available packets, this exception can be thrown on decoding errors.

**Returns**

0 if no data is available, number of packets otherwise.

**property id:** str

**initialize**(*args: any | None = None*) → any

Perform initializations step which can not be done in constructor or which require returnvalues.

**is\_open**() → bool

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**open**(*args: any | None = None*)

Opens the communication interface to allow communication.

**Returns**

**receive**(*poll\_timeout: float = 0*) → List[bytes]

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

**Parameters**

**parameters**

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

**Returns**

**send**(*data: bytes*)

Send raw data.

**Raises**

**SendError** – Sending failed for some reason.

## Serial Communication with COBS Module

**class** tmtccmd.com.serial\_cobs.SerialCobsComIF(*ser\_cfg*: SerialCfg)

Bases: *SerialComBase*, *ComInterface*

Serial communication interface which uses the [COBS protocol](#) to encode and decode packets.

This class will spin up a receiver thread on the [open\(\)](#) call to poll for COBS encoded packets. This means that the [close\(\)](#) call might block until the receiver thread has shut down.

**close**(*args*: any | *None* = *None*) → *None*

Closes the ComIF and releases any held resources (for example a Communication Port).

### Returns

**data\_available**(*timeout*: float, *parameters*: any = 0) → int

Check whether TM packets are available.

### Parameters

- **timeout** – Can be used to block on available data if supported by the specific communication interface.
- **parameters** – Can be an arbitrary parameter.

### Raises

[ReceptionDecodeError](#) – If the underlying COM interface uses encoding and decoding when determining the number of available packets, this exception can be thrown on decoding errors.

### Returns

0 if no data is available, number of packets otherwise.

**property id**: str

**initialize**(*args*: any | *None* = *None*) → any

Perform initializations step which can not be done in constructor or which require returnvalues.

**is\_open**() → bool

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**open**(*args*: any | *None* = *None*) → *None*

Spins up a receiver thread to permanently check for new COBS encoded packets.

**receive**(*parameters*: any = 0) → List[bytes]

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

### Parameters

**parameters**

### Raises

[ReceptionDecodeError](#) – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

### Returns

**send**(*data*: bytes)

Send raw data.

**Raises**

**`SendError`** – Sending failed for some reason.

**Serial Communication with DLE Module**

```
class tmtccmd.com.serial_dle.DleCfg(dle_queue_len: int | None = None, dle_max_frame: int | None = None, encode_cr: bool = True)
```

Bases: `object`

**`dle_max_frame`**: `int` | `None` = `None`

**`dle_queue_len`**: `int` | `None` = `None`

**`encode_cr`**: `bool` = `True`

```
class tmtccmd.com.serial_dle.SerialDleComIF(ser_cfg: SerialCfg, dle_cfg: DleCfg | None)
```

Bases: `SerialComBase`, `ComInterface`

Serial communication interface which uses the `DLE protocol` to encode and decode packets.

This class will spin up a receiver thread on the `open()` call to poll for DLE encoded packets. This means that the `close()` call might block until the receiver thread has shut down.

**`close`**(args: any | `None` = `None`) → `None`

Closes the ComIF and releases any held resources (for example a Communication Port).

**Returns**

**`data_available`**(timeout: float, parameters: any = 0) → `int`

Check whether TM packets are available.

**Parameters**

- **`timeout`** – Can be used to block on available data if supported by the specific communication interface.
- **`parameters`** – Can be an arbitrary parameter.

**Raises**

**`ReceptionDecodeError`** – If the underlying COM interface uses encoding and decoding when determining the number of available packets, this exception can be thrown on decoding errors.

**Returns**

0 if no data is available, number of packets otherwise.

**property id**: `str`

**`initialize`**(args: any | `None` = `None`) → any

Perform initializations step which can not be done in constructor or which require returnvalues.

**`is_open`**() → `bool`

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**`open`**(args: any | `None` = `None`) → `None`

Spins up a receiver thread to permanently check for new DLE encoded packets.

**receive**(*parameters: any = 0*) → List[bytes]

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

**Parameters**

**parameters**

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

**Returns**

**send**(*data: bytes*)

Send raw data.

**Raises**

**SendError** – Sending failed for some reason.

## Generic Serial Modules

```
class tmtccmd.com.serial_base.SerialCfg(com_if_id: str, serial_port: str, baud_rate: int, serial_timeout: float)
```

Bases: `object`

**baud\_rate**: `int`

**com\_if\_id**: `str`

**serial\_port**: `str`

**serial\_timeout**: `float`

```
class tmtccmd.com.serial_base.SerialComBase(logger: Logger, ser_cfg: SerialCfg, ser_com_type: SerialCommunicationType)
```

Bases: `object`

**close\_port**()

**static data\_available\_from\_queue**(*timeout: float, reception\_buffer: deque*)

**is\_port\_open**() → `bool`

**open\_port**()

```
class tmtccmd.com.serial_base.SerialCommunicationType(value)
```

Bases: `Enum`

Right now, two serial communication methods are supported. One uses frames with a fixed size containing PUS packets and the other uses a simple ASCII based transport layer called DLE. If DLE is used, it is expected that the sender side encoded the packets with the DLE protocol. Any packets sent will also be encoded.

**COBS** = 0

**DLE\_ENCODING** = 2

```
class tmtccmd.com.serial_base.SerialConfigIds(value)
```

Bases: [Enum](#)

An enumeration.

```
SERIAL_BAUD_RATE = 2
```

```
SERIAL_COMM_TYPE = 4
```

```
SERIAL_DLE_MAX_FRAME_SIZE = 7
```

```
SERIAL_DLE_QUEUE_LEN = 6
```

```
SERIAL_FRAME_SIZE = 5
```

```
SERIAL_PORT = 1
```

```
SERIAL_TIMEOUT = 3
```

```
tmtccmd.com.ser_utils.check_port_validity(com_port_to_check: str) → bool
```

```
tmtccmd.com.ser_utils.determine_baud_rate(json_cfg_path: str) → int
```

Determine baud rate. Tries to read from JSON first. If the baud rate is not contained in the config JSON, prompt it from user instead with the option to store value in JSON file.

#### Returns

Determined baud rate

```
tmtccmd.com.ser_utils.determine_com_port(json_cfg_path: str) → str
```

Determine serial port. Tries to read from JSON first. If the serial port is not contained in the config JSON, prompt it from user instead with the option to store value in JSON file.

#### Returns

Determined serial port

```
tmtccmd.com.ser_utils.find_com_port_from_hint(hint: str) → Tuple[bool, str]
```

Find a COM port based on a hint string

```
tmtccmd.com.ser_utils.prompt_com_port() → str
```

## Dummy Module

Dummy Virtual Communication Interface. Currently serves to use the TMTC program without needing external hardware or an extra socket

```
class tmtccmd.com.dummy.DummyComIF
```

Bases: [ComInterface](#)

```
close(args: any | None = None) → None
```

Closes the ComIF and releases any held resources (for example a Communication Port).

#### Returns

```
data_available(timeout: float = 0, parameters: any = 0)
```

Check whether TM packets are available.

#### Parameters

- **timeout** – Can be used to block on available data if supported by the specific communication interface.

- **parameters** – Can be an arbitrary parameter.

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding when determining the number of available packets, this exception can be thrown on decoding errors.

**Returns**

0 if no data is available, number of packets otherwise.

**property id:** `str`

**initialize**(args: any | None = None) → any

Perform initializations step which can not be done in constructor or which require returnvalues.

**is\_open**() → bool

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**open**(args: any | None = None) → None

Opens the communication interface to allow communication.

**Returns**

**receive**(parameters: any = 0) → List[bytes]

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

**Parameters**

**parameters**

**Raises**

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

**Returns**

**send**(data: bytes)

Send raw data.

**Raises**

**SendError** – Sending failed for some reason.

**class** tmtccmd.com.dummy.DummyHandler

Bases: `object`

**generate\_reply\_package**()

Generate a reply package. Currently, this only generates a reply for a ping telecommand.

**insert\_telecommand**(data: bytes)

**pass\_telecommand**(data: bytearray)

Deprecated since version 6.0.0: Use insert\_telecommand instead

**receive\_reply\_package**() → List[bytes]

## Utilities Module

`tmtccmd.com.utils.determine_com_if`(*com\_if\_dict*: *Mapping[str, Tuple[str, Any]]*, *json\_cfg\_path*: *str*, *use\_prompts*: *bool*) → *str*

`tmtccmd.com.utils.prompt_com_if`(*com\_if\_dict*: *Mapping[str, Tuple[str, Any]]*) → *str*

`tmtccmd.com.utils.store_com_if_json`(*com\_if\_string*: *str*, *json\_cfg\_path*: *str*)

## QEMU Module

QEMU\_SERIAL Communication Interface to communicate with emulated QEMU\_SERIAL hardware via the UART interface.

It utilizes the the asyncio library.

### Requirements:

Python >= 3.7 (asyncio support)

### Instructions:

Run QEMU\_SERIAL (modified for OBSW) via

```
qemu-system-arm -M isis-obc -monitor stdio -bios path/to/sourceobsw-at91sam9g20_ek-sdram.bin -qmp
unix:/tmp/qemu,server -S
```

Then run the telecommand script with -c 2

**class** `tmtccmd.com.qemu.DataFrame`(*seq*, *cat*, *frame\_id*, *data=None*)

Bases: `object`

Basic protocol unit for communication via the IOX API introduced for external device emulation

**bytes**()

Convert this protocol unit to raw bytes

**class** `tmtccmd.com.qemu.QEMUComIF`(*serial\_cfg*: `SerialCfg`, *ser\_com\_type*: `SerialCommunicationType` = `SerialCommunicationType.DLE_ENCODING`)

Bases: `ComInterface`

Specific Communication Interface implementation of the QEMU\_SERIAL USART protocol for the TMTC software

**close**(*\_=None*) → `None`

Closes the ComIF and releases any held resources (for example a Communication Port).

### Returns

**data\_available**(*timeout*: *float* = 0, *\_*=0) → `int`

Check whether TM packets are available.

### Parameters

- **timeout** – Can be used to block on available data if supported by the specific communication interface.
- **parameters** – Can be an arbitrary parameter.

### Raises

`ReceptionDecodeError` – If the underlying COM interface uses encoding and decoding



when determining the number of available packets, this exception can be thrown on decoding errors.

#### Returns

0 if no data is available, number of packets otherwise.

**data\_available\_dle**(*timeout: float = 0*) → *int*

**data\_available\_fixed\_frame**(*timeout: float = 0*) → *int*

**property id:** *str*

**initialize**(*\_ = None*)

Needs to be called by application code once for DLE mode!

**is\_open**() → *bool*

Can be used to check whether the communication interface is open. This is useful if opening a COM interface takes a longer time and is non-blocking

**open**(*\_ = None*) → *None*

Opens the communication interface to allow communication.

#### Returns

**async poll\_dle\_packets**()

**receive**(*\_*) → *List[bytes]*

Returns a list of received packets. The child class can use a separate thread to poll for the packets or use some other mechanism and container like a deque to store packets to be returned here.

#### Parameters

**parameters**

#### Raises

**ReceptionDecodeError** – If the underlying COM interface uses encoding and decoding and the decoding fails, this exception will be returned.

#### Returns

**send**(*data: bytearray*)

Send raw data.

#### Raises

**SendError** – Sending failed for some reason.

**send\_data**(*data: bytearray*)

**async send\_data\_async**(*data*)

**set\_dle\_settings**(*dle\_queue\_len: int, dle\_max\_frame: int, dle\_timeout: float*)

**set\_fixed\_frame\_settings**(*serial\_frame\_size: int*)

**async start\_dle\_polling**()

**class** tmtccmd.com.qemu.QmpConnection(*addr='/tmp/qemu'*)

Bases: *object*

A connection to a QEMU\_SERIAL machine via QMP

**close()**

Close this connection

**async cont()**

Continue machine execution if it has been paused

**async open()**

Open this connection. Connect to the machine ensure that the connection is ready to use after this call.

**async quit()**

Quit the emulation. This causes the emulator to (non-gracefully) shut down and close.

**async stop()**

Stop/pause machine execution

**exception** tmtccmd.com.qemu.QmpException(*ret*, \**args*, \*\**kwargs*)Bases: [Exception](#)

An exception caused by the QML/QEMU\_SERIAL as response to a failed command

**class** tmtccmd.com.qemu.QmpProtocol(*conn*)Bases: [Protocol](#)

The QMP transport protocol implementation

**connection\_lost**(*exc*)

Called when the connection is lost or closed.

The argument is an exception object or None (the latter meaning a regular EOF is received or the connection was aborted or closed).

**connection\_made**(*transport*)

Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for `data_received()` calls. When the connection is closed, `connection_lost()` is called.**data\_received**(*data*)

Called when some data is received.

The argument is a bytes object.

**class** tmtccmd.com.qemu.Usart(*addr*)Bases: [object](#)**close()**

Close this connection

**async static create\_async**(*addr*)**flush()****get\_data\_in\_waiting**() → [int](#)**inject\_frame\_error**()

Inject a frame error (set CSR\_FRAME)

**inject\_overrun\_error**()

Inject an overrun error (set CSR\_OVRE)

**inject\_parity\_error()**

Inject a parity error (set CSR\_PARE)

**inject\_timeout\_error()**

Inject a timeout (set CSR\_TIMEOUT)

**new\_data\_available()** → bool**async open()**

Open this connection

**read(*n*)**

Wait for 'n' bytes to be received from the USART timeout in seconds

**async read\_async(*n*, *timeout=None*)**

Wait for 'n' bytes to be received from the USART.

This function will return early if the specified timeout (in seconds) is exceeded. In this case, only the data received up to that point will be returned. If timeout is None, no timeout will be set.

**async read\_until\_async(*expected*, *size=None*, *timeout=None*)**

Read data until either the expected byte sequence has been found, the specified number of bytes has been received, or the timeout has occurred.

This function will return whatever data has been received up until the first termination condition has been met. In case size is None, there will be no size limit. In case timeout is None, there will be no timeout.

**async write(*data*)**

Write data (bytes) to the USART device

**class tmtccmd.com.qemu.UsartProtocol(*conn*)**

Bases: Protocol

The USART transport protocol implementation

**connection\_lost(*exc*)**

Called when the connection is lost or closed.

The argument is an exception object or None (the latter meaning a regular EOF is received or the connection was aborted or closed).

**connection\_made(*transport*)**

Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data\_received() calls. When the connection is closed, connection\_lost() is called.

**data\_received(*data*)**

Called when some data is received.

The argument is a bytes object.

**exception tmtccmd.com.qemu.UsartStatusException(*errn*, \**args*, \*\**kwargs*)**

Bases: Exception

An exception returned by the USART send command

**tmtccmd.com.qemu.parse\_dataframes(*buf*)**

Parse a variable number of DataFrames from the given byte buffer

**tmtccmd.com.qemu.start\_background\_loop(*loop: AbstractEventLoop*)** → None

## 1.5.2 ECSS & CCSDS Package

This module contains PUS data structures and helpers common for both PUS telemetry and telecommands.

Content:

- `tmtccmd.pus.VerificationWrapper` helper class

### PUS Package

#### Submodules

#### Module contents

This module contains PUS data structures and helpers common for both PUS telemetry and telecommands.

Content:

- `tmtccmd.pus.VerificationWrapper` helper class

**class** `tmtccmd.pus.CustomFsfwPusService(value)`

Bases: `IntEnum`

An enumeration.

**SERVICE\_200\_MODE = 200**

**class** `tmtccmd.pus.VerificationWrapper(pus_verificator: PusVerificator, console_logger: Logger | None, file_logger: Logger | None)`

Bases: `object`

**add\_tc**(*pus\_tc: PusTc*) → `bool`

**add\_tm**(*srv\_l\_tm: ServiceITm*) → `TmCheckResult`

**dlog**(*log\_str: str*, *level: int* = 20)

**log\_progress\_to\_console\_from\_status**(*status: VerificationStatus*, *req\_id: RequestId*, *subservice: Subservice* | *None* = *None*)

**log\_to\_console**(*srv\_l\_tm: ServiceITm*, *res: TmCheckResult*)

**log\_to\_console\_from\_req\_id**(*req\_id: RequestId*, *res: TmCheckResult*, *subservice: Subservice* | *None* = *None*)

**log\_to\_file**(*srv\_l\_tm: ServiceITm*, *res: TmCheckResult*)

**log\_to\_file\_from\_req\_id**(*req\_id: RequestId*, *res: TmCheckResult*, *subservice: Subservice* | *None* = *None*)

**log\_to\_file\_from\_status**(*status: VerificationStatus*, *req\_id: RequestId*, *subservice: Subservice* | *None* = *None*)

**static step\_num**(*status: VerificationStatus*)

**property verificator:** `PusVerificator`

`tmtccmd.pus.cross_mark_unicode(with_color: bool)` → `str`

```

tmtccmd.pus.dash_unicode(with_color: bool) → str
tmtccmd.pus.gen_console_char_from_status(status: StatusField, with_color: bool)
tmtccmd.pus.gen_file_char_from_status(status: StatusField)
tmtccmd.pus.tick_mark_unicode(with_color: bool) → str

```

### Service 1 Telecommand Verification Module

```

class tmtccmd.pus.tm.s1_verification.Service1FsfwWrapper(tm: ServiceITm)
    Bases: object

```

### Service 2 Raw Commanding Telemetry Module

### Service 3 Housekeeping Modules

PUS Service 3 components

```

class tmtccmd.pus.tm.s3_hk_base.HkContentType(value)
    Bases: Enum

```

An enumeration.

```

DEFINITIONS = <class 'enum.auto'>

```

```

HK = <class 'enum.auto'>

```

```

class tmtccmd.pus.tm.s3_hk_base.Service3Base(object_id: int, custom_hk_handling: bool = False)
    Bases: object

```

Base class. The TMTC core provides a Service 3 implementation which is intended to be used with the FSFW. However, users can define an own Service 3 implementation.

The TMTC printer utility uses the fields defined in this base class to perform printouts so if a custom class is defined, the user should implement this class and fill the fields in the TM handling hook if printout of the HK field and validity checking is desired.

```

property has_custom_hk_handling: bool

```

```

property hk_definitions_list: Tuple[List, List]

```

Can be implemented by a child class to print definitions lists. The first list should contain a header with parameter names, and the second list shall contain the corresponding set IDs

```

property object_id: ComponentIdU32

```

```

property set_id: int

```

Contains definitions and functions related to PUS Service 3 Telecommands.

```

tmtccmd.pus.tc.s3_fsfw_hk.create_disable_periodic_hk_command(sid: bytes) → PusTc

```

```

tmtccmd.pus.tc.s3_fsfw_hk.create_disable_periodic_hk_command_with_diag(diag: bool, sid: bytes)
    → PusTc

```

Deprecated since version v6.0.0rc0: use diagnostic agnostic API if possible

`tmtccmd.pus.tc.s3_fsfw_hk.create_enable_periodic_hk_command(sid: bytes) → PusTc`

`tmtccmd.pus.tc.s3_fsfw_hk.create_enable_periodic_hk_command_with_diag(diag: bool, sid: bytes) → PusTc`

Deprecated since version v6.0.0rc0: use diagnostic agnostic API if possible

`tmtccmd.pus.tc.s3_fsfw_hk.create_enable_periodic_hk_command_with_interval(sid: bytes, interval_seconds: float) → Tuple[PusTc, PusTc]`

`tmtccmd.pus.tc.s3_fsfw_hk.create_enable_periodic_hk_command_with_interval_with_diag(diag: bool, sid: bytes, interval_seconds: float) → Tuple[PusTc, PusTc]`

Deprecated since version v6.0.0rc0: use diagnostic agnostic API if possible

`tmtccmd.pus.tc.s3_fsfw_hk.create_modify_collection_interval_cmd(sid: bytes, interval_seconds: float) → PusTc`

`tmtccmd.pus.tc.s3_fsfw_hk.create_modify_collection_interval_cmd_with_diag(diag: bool, sid: bytes, interval_seconds: float) → PusTc`

Deprecated since version v6.0.0rc0: use diagnostic agnostic API if possible

`tmtccmd.pus.tc.s3_fsfw_hk.create_request_one_diag_command(sid: bytes) → PusTc`

`tmtccmd.pus.tc.s3_fsfw_hk.create_request_one_hk_command(sid: bytes) → PusTc`

`tmtccmd.pus.tc.s3_fsfw_hk.disable_periodic_hk_command(diag: bool, sid: bytes) → PusTc`

Deprecated since version v4.0.0a2: use create... API instead

`tmtccmd.pus.tc.s3_fsfw_hk.enable_periodic_hk_command(diag: bool, sid: bytes) → PusTc`

Deprecated since version v4.0.0a2: use create... API instead

`tmtccmd.pus.tc.s3_fsfw_hk.enable_periodic_hk_command_with_interval(diag: bool, sid: bytes, interval_seconds: float) → Tuple[PusTc, PusTc]`

Deprecated since version v4.0.0a2: use create... API instead

`tmtccmd.pus.tc.s3_fsfw_hk.generate_one_diag_command(sid: bytes) → PusTc`

Deprecated since version v4.0.0a2: use create... API instead

`tmtccmd.pus.tc.s3_fsfw_hk.generate_one_hk_command(sid: bytes) → PusTc`

Deprecated since version v4.0.0a2: use create... API instead

`tmtccmd.pus.tc.s3_fsfw_hk.make_interval(interval_seconds: float) → bytearray`

`tmtccmd.pus.tc.s3_fsfw_hk.make_sid(object_id: bytes, set_id: int) → bytearray`

`tmtccmd.pus.tc.s3_fsfw_hk.modify_collection_interval(diag: bool, sid: bytes, interval_seconds: float) → PusTc`

Deprecated since version v4.0.0a2: use create... API instead

## Service 5 Event Module

Contains definitions and functions related to PUS Service 5 Telecommands.

`tmtccmd.pus.tc.s5_event.create_disable_event_reporting_command(apid: int = 0, seq_count: int = 0) → PusTc`

`tmtccmd.pus.tc.s5_event.create_enable_event_reporting_command(apid: int = 0, seq_count: int = 0) → PusTc`

`tmtccmd.pus.tc.s5_event.pack_disable_event_reporting_command(apid: int = 0, seq_count: int = 0) → PusTc`

Deprecated since version v4.0.0a0: use create... API instead

`tmtccmd.pus.tc.s5_event.pack_enable_event_reporting_command(apid: int = 0, seq_count: int = 0) → PusTc`

Deprecated since version v4.0.0a0: use create... API instead

## Service 5 Event Module - sat-rs support

sat-rs specific PUS event support.

**class** `tmtccmd.pus.s5_satrs_event_defs.EventSeverity(value)`

Bases: `IntEnum`

An enumeration.

**HIGH** = 3

**INFO** = 0

**LOW** = 1

**MEDIUM** = 2

**class** `tmtccmd.pus.s5_satrs_event_defs.EventU32(severity: 'EventSeverity', group_id: 'int', unique_id: 'int')`

Bases: `object`

**group\_id**: `int`

**severity**: `EventSeverity`

**unique\_id**: `int`

**classmethod** `unpack(data: bytes) → EventU32`

## Service 5 Event Module - FSFW support

FSFW specific PUS event support

```
class tmtccmd.pus.s5_fsfw_event_defs.EventInfo(id: int = 0, name: str = "", severity: str = "", info: str = "", file_location: str = "")
```

Bases: `object`

`file_location: str = ''`

`id: int = 0`

`info: str = ''`

`name: str = ''`

`severity: str = ''`

```
class tmtccmd.pus.s5_fsfw_event_defs.Severity(value)
```

Bases: `IntEnum`

An enumeration.

`HIGH = 4`

`INFO = 1`

`LOW = 2`

`MEDIUM = 3`

```
tmtccmd.pus.s5_fsfw_event_defs.str_to_severity(string: str) → Severity | None
```

Contains classes and functions to deserialize PUS Service 5 Telemetry

```
class tmtccmd.pus.tm.s5_fsfw_event.EventDefinition(event_id: 'int', reporter_id: 'bytes', param1: 'int', param2: 'int')
```

Bases: `object`

`classmethod empty() → EventDefinition`

`event_id: int`

`classmethod from_bytes(data: bytes) → EventDefinition`

`pack() → bytes`

`param1: int`

`param2: int`

`reporter_id: bytes`

```
class tmtccmd.pus.tm.s5_fsfw_event.Service5Tm(apid: int, subservice: Subservice, event: EventDefinition, timestamp: bytes, ssc: int = 0, packet_version: int = 0, space_time_ref: int = 0, destination_id: int = 0)
```

Bases: `AbstractPusTm`

`property ccsds_version: int`



```

property event_definition: EventDefinition
classmethod from_tm(pus_tm: PusTm) → Service5Tm
pack() → bytearray
property packet_id: PacketId
property packet_seq_control: PacketSeqCtrl
property service: int
property severity: Severity
property source_data: bytes
property sp_header: SpacePacketHeader
property subservice: int
property timestamp: bytes
classmethod unpack(data: bytes, timestamp_len: int) → Service5Tm

```

## Service 8 Action Commanding Module

FSFW specific PUS actions support

```

class tmtccmd.pus.s8_fsfw_action_defs.CustomSubservice(value)
    Bases: IntEnum
    An enumeration.
    TC_FUNCTIONAL_CMD = 128
    TM_DATA_REPLY = 130

```

Contains classes and functions to handle PUS Service 8 telemetry.

```

tmtccmd.pus.tc.s8_fsfw_action.create_action_cmd(object_id: bytes, action_id: int, user_data: bytes =
                                                b'', apid: int = 0, seq_count: int = 0) → PusTc

tmtccmd.pus.tc.s8_fsfw_action.make_action_id(action_id: int) → bytearray

tmtccmd.pus.tc.s8_fsfw_action.make_fsfw_action_cmd(object_id: bytes, action_id: int, user_data: bytes
                                                = b'', apid: int = 0, seq_count: int = 0) → PusTc

```

Deprecated since version v4.0.0a2: use create... API instead

## Service 11 Telecommand Scheduling Module

```

class tmtccmd.pus.s11_tc_sched_defs.Subservice(value)
    Bases: IntEnum
    Unless specified, TCs and TMs are related to a request ID
    TC_DELETE = 5

```

```
TC_DELETE_WITH_FILTER = 6
TC_DETAIL_REPORT_FILTER_BASED = 11
TC_DETAIL_REPORT_TIME_BASED = 9
TC_DISABLE = 2
TC_ENABLE = 1
TC_INSERT = 4
TC_RESET = 3
TC_TIMEShift = 7
TC_TIMEShift_ALL = 15
TC_TIMEShift_WITH_FILTER = 8
TM_DETAIL_REPORT_FILTER_BASED = 12
TM_DETAIL_REPORT_TIME_BASED = 10
```

```
class tmtccmd.pus.s11_tc_sched_defs.TcSchedReqId(apid: int, seq_cnt: int, src_id: int)
```

```
    Bases: object
```

```
    classmethod build_from_tc(tc: PusTc) → TcSchedReqId
```

```
    property id_u64
```

```
    pack() → bytes
```

```
class tmtccmd.pus.s11_tc_sched_defs.TypeOfTimeWindow(value)
```

```
    Bases: IntEnum
```

```
    An enumeration.
```

```
    FROM_TIMETAG = 2
```

```
    FROM_TIMETAG_TO_TIMETAG = 1
```

```
    SELECT_ALL = 0
```

```
    TO_TIMETAG = 3
```

## Service 17 Test Module

```
class tmtccmd.pus.s17_test_defs.CustomSubservice(value)
```

```
    Bases: IntEnum
```

```
    An enumeration.
```

```
    TC_GEN_EVENT = 128
```

## Service 20 Parameter Module - FSFW support

```
class tmtccmd.pus.s20_fsfw_param_defs.CustomSubservice(value)
```

Bases: `IntEnum`

An enumeration.

**TC\_DUMP** = 129

**TC\_LOAD** = 128

**TM\_DUMP\_REPLY** = 130

```
class tmtccmd.pus.s20_fsfw_param_defs.FsfwParamId(object_id: bytes, param_id: ParameterId, ptc: Ptc,
                                                  pfc: int, rows: int, columns: int)
```

Bases: `object`

Wrapper for the whole FSFW specific parameter data. It contains the ECSS PTC and PFC numbers and the number of columns and rows in the parameter. See <https://ecss.nl/standard/ecss-e-st-70-41c-space-engineering-telemetry-and-telecommand-packet-utilization-15-april-2016/> p.428 for more information.

### Parameters

- **ptc** – ECSS PTC number
- **pfc** – ECSS PFC number
- **rows** – Number of rows in parameter (for matrix entries, 1 for vector entries, 1 for scalar entries)
- **columns** – Number of columns in parameter (for matrix or vector entries, 1 for scalar entries)

### Returns

Parameter information field as 4 byte bytearray

**columns:** `int`

**object\_id:** `bytes`

**pack()** → `bytearray`

Convert the wrapper to the raw byte format expected for PUS TC or PUS TM creation.

**param\_id:** `ParameterId`

**pfc:** `int`

**ptc:** `Ptc`

**rows:** `int`

**classmethod unpack**(data: `bytes`) → `FsfwParamId`

```
class tmtccmd.pus.s20_fsfw_param_defs.Parameter(fsfw_param_id: 'FsfwParamId', param_raw: 'bytes')
```

Bases: `object`

**property columns**

**classmethod empty**()

**fsfw\_param\_id:** `FsfwParamId`

**property** object\_id

**pack()** → bytearray

Convert the wrapper to the raw byte format expected for PUS TC or PUS TM creation.

**property** param\_id

**param\_raw:** bytes

**parse\_scalar\_param()** → int | float

**property** pfc

**property** ptc

**property** rows

**classmethod** **unpack**(data: bytes) → Parameter

**class** tmtccmd.pus.s20\_fsfw\_param\_defs.ParameterId(domain\_id: 'int', unique\_id: 'int', linear\_index: 'int')

Bases: object

**as\_u32()** → int

**domain\_id:** int

**classmethod** **empty()** → ParameterId

**linear\_index:** int

**pack()** → bytes

**unique\_id:** int

**classmethod** **unpack**(data: bytes) → ParameterId

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_matrix\_double\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameters: Sequence[Sequence[float]]) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_matrix\_float\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameters: Sequence[Sequence[float]]) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_boolean\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: bool) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_double\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: float) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_float\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: float) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_i16\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: int) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_i32\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: int) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_i8\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: int) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_u16\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: int) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_u32\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: int) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_scalar\_u8\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: int) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_vector\_double\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameters: Sequence[float]) → Parameter

tmtccmd.pus.s20\_fsfw\_param\_defs.create\_vector\_float\_parameter(object\_id: bytes, domain\_id: int, unique\_id: int, parameters: Sequence[float])

tmtccmd.pus.s20\_fsfw\_param\_defs.deserialize\_scalar\_entry(ptc: int, pfc: int, param\_data: bytes) → int | float

Try to deserialize a scalar parameter entry (row = 1 and column = 1).

#### Raises

- **ValueError** – Passed parameter data length invalid.
- **NotImplementedError** – Parsing of parameter data not implemented for given PTC and PFC.

tmtccmd.pus.s20\_fsfw\_param\_defs.parse\_scalar\_param(wrapper: Parameter) → int | float

Contains definitions and functions related to PUS Service 20 Telecommands.

tmtccmd.pus.tc.s20\_fsfw\_param.create\_dump\_param\_cmd(param\_fsfw\_id: FsfwParamId, apid: int = 0) → PusTc

tmtccmd.pus.tc.s20\_fsfw\_param.create\_load\_param\_cmd(parameter: Parameter, apid: int = 0) → PusTc

tmtccmd.pus.tc.s20\_fsfw\_param.create\_load\_param\_cmd\_from\_raw(parameter\_raw: bytes, apid: int = 0) → PusTc

tmtccmd.pus.tc.s20\_fsfw\_param.pack\_boolean\_parameter\_app\_data(object\_id: bytes, domain\_id: int, unique\_id: int, parameter: bool) → bytearray | None

Deprecated since version 3.1.0: use `create_scalar_boolean_parameter` instead

```
tmtccmd.pus.tc.s20_fsfw_param.pack_parameter_id(domain_id: int, unique_id: int, linear_index: int) →  
                                             bytearray
```

Packs the Parameter ID (bytearray with 4 bytes) which is part of the service 20 packets. The first byte of the parameter ID is the domain ID, the second byte is a unique ID and the last two bytes are a linear index if a parameter is not loaded from index 0. :param domain\_id: One byte domain ID :param unique\_id: One byte unique ID :param linear\_index: Two byte linear index.

Deprecated since version 4.0.0a2: use `ParameterId` helper class with `pack()` instead

```
tmtccmd.pus.tc.s20_fsfw_param.pack_scalar_boolean_parameter_app_data(object_id: bytes,  
                                                                    domain_id: int,  
                                                                    unique_id: int, parameter:  
                                                                    bool) → bytearray | None
```

Tailored towards FSFW applications.

#### Parameters

- `object_id`
- `domain_id`
- `unique_id`
- `parameter`

#### Returns

Application data

Deprecated since version 4.0.0a2: Please use `create_scalar_boolean_parameter` instead

```
tmtccmd.pus.tc.s20_fsfw_param.pack_scalar_double_param_app_data(object_id: bytes, domain_id: int,  
                                                                unique_id: int, parameter: float)  
                                                                → bytearray | None
```

Deprecated since version 4.0.0a2: Please use `create_scalar_double_parameter` instead

```
tmtccmd.pus.tc.s20_fsfw_param.pack_scalar_float_param_app_data(object_id: bytes, domain_id: int,  
                                                                unique_id: int, parameter: float)  
                                                                → bytearray | None
```

Deprecated since version 4.0.0a2: use `create_scalar_float_parameter` instead

```
tmtccmd.pus.tc.s20_fsfw_param.pack_scalar_u8_parameter_app_data(object_id: bytes, domain_id: int,  
                                                                unique_id: int, parameter: int)  
                                                                → bytearray | None
```

Tailored towards FSFW applications.

#### Parameters

- `object_id`
- `domain_id`
- `unique_id`
- `parameter`

#### Returns

Application data

Deprecated since version 4.0.0a2: Please use `create_scalar_u8_parameter` instead

`tmtccmd.pus.tc.s20_fsfw_param.pack_type_and_matrix_data`(*ptc: int, pfc: int, rows: int, columns: int*)  
→ `bytearray`

Deprecated since version 4.0.0a2: use ParamWrapper helper class with `pack()` instead

`tmtccmd.pus.tc.s20_fsfw_param.prepare_param_packet_header`(*object\_id: bytes, domain\_id: int, unique\_id: int, ptc: Ptc, pfc: int, rows: int, columns: int, start\_at\_idx: int = 0*)  
→ `bytearray` | `None`

Deprecated since version 4.0.0a2: use ParamWrapper helper class with `pack()` instead

**class** `tmtccmd.pus.tm.s20_fsfw_param.Service20FsfwTm`(*subservice: int, source\_data: bytes, timestamp: bytes, apid: int = 0*)

Bases: `AbstractPusTm`

**property** `ccsds_version: int`

**classmethod** `empty()` → `Service20FsfwTm`

**classmethod** `from_tm(tm: PusTm)`

**property** `object_id: bytes`

**pack()** → `bytearray`

**property** `packet_id: PacketId`

**property** `packet_seq_control: PacketSeqCtrl`

**property** `service: int`

**property** `source_data: bytes`

**property** `sp_header: SpacePacketHeader`

**property** `subservice: int`

**property** `timestamp: bytes`

**classmethod** `unpack(raw_telemetry: bytes, timestamp_len: int)` → `Service20FsfwTm`

**class** `tmtccmd.pus.tm.s20_fsfw_param.Service20ParamDumpWrapper`(*param\_tm: Service20FsfwTm*)

Bases: `object`

**property** `base_tm: PusTm`

**get\_param()** → `Parameter`

Tries to build a `Parameter` from the own raw telemetry data.

**Raises**

**ValueError** – Telemetry source data too short.

## Service 200 Mode Commanding Module - FSFW support

```
class tmtccmd.pus.s200_fsfw_mode_defs.Subservice(value)
```

Bases: `IntEnum`

An enumeration.

```
TC_MODE_ANNOUNCE = 4
```

```
TC_MODE_ANNOUNCE_RECURSIVE = 5
```

```
TC_MODE_COMMAND = 1
```

```
TC_MODE_COMMAND_FORCES = 2
```

```
TC_MODE_READ = 3
```

```
TM_CANT_REACH_MODE = 7
```

```
TM_MODE_REPLY = 6
```

```
TM_WRONG_MODE_REPLY = 8
```

Core components for mode commanding (custom PUS service).

```
class tmtccmd.pus.tc.s200_fsfw_mode.Mode(value)
```

Bases: `IntEnum`

Standard modes when commanding objects. These mode IDs are reserved by the FSFW, so it is recommended to avoid these numbers for custom modes.

```
NORMAL = 2
```

```
OFF = 0
```

```
ON = 1
```

```
RAW = 3
```

```
tmtccmd.pus.tc.s200_fsfw_mode.create_announce_mode_command(object_id: bytes) → PusTc
```

```
tmtccmd.pus.tc.s200_fsfw_mode.create_announce_mode_recursive_command(object_id: bytes) →  
PusTc
```

```
tmtccmd.pus.tc.s200_fsfw_mode.create_mode_command(object_id: bytes, mode: int | Mode, submode: int)  
→ PusTc
```

```
tmtccmd.pus.tc.s200_fsfw_mode.create_read_mode_command(object_id: bytes) → PusTc
```

```
tmtccmd.pus.tc.s200_fsfw_mode.pack_mode_command(object_id: bytes, mode: int | Mode, submode: int) →  
PusTc
```

Deprecated since version v4.0.0a2: use create... API instead

```
tmtccmd.pus.tc.s200_fsfw_mode.pack_mode_data(object_id: bytes, mode: Mode | int, submode: int) →  
bytearray
```

FSFW modes: Mode 0: Off, Mode 1: Mode On, Mode 2: Mode Normal, Mode 3: Mode Raw

Base class for Service 200 mode commanding reply handling.



```
class tmtccmd.pus.tm.s200_fsfw_mode.Service200FsfwReader(tm: PusTm)
```

Bases: *object*

**contains\_mode()** → *bool*

**is\_cant\_reach\_mode\_reply()** → *bool*

```
class tmtccmd.pus.tm.s200_fsfw_mode.Service200FsfwTm(*args, **kwargs)
```

Bases: *PusTmBase*, *PusTmInfoBase*

Deprecated since version 7.0.0: use Service200FsfwReader instead

**append\_telemetry\_column\_headers**(*header\_list: list*)

Default implementation adds the PUS header content header (confusing, I know) to the list which can then be printed with a simple print() command. To add additional headers, override this method. Any child class should call this function as well if header information is required.

**Parameters**

**header\_list** – Header content will be appended to this list

**Returns**

**append\_telemetry\_content**(*content\_list: list*)

Default implementation adds the PUS header content to the list which can then be printed with a simple print() command. To add additional content, override this method. Any child class should call this function as well if header information is required.

**Parameters**

**content\_list** – Header content will be appended to this list

**Returns**

**classmethod unpack**(*raw\_telemetry: bytes*, *time\_reader: CcsdsTimeProvider | None*) → *Service200FsfwTm*

## Service 201 Health Commanding Module - FSFW support

```
class tmtccmd.pus.s201_fsfw_health_defs.Subservice(value)
```

Bases: *IntEnum*

An enumeration.

**TC\_ANNOUNCE\_HEALTH** = 3

**TC\_ANNOUNCE\_HEALTH\_ALL** = 4

**TC\_SET\_HEALTH** = 1

**TM\_HEALTH\_SET** = 2

## CFDP Module

### Request Submodule

```
class tmtccmd.cfdp.request.CfdpRequestBase(req_type: CfdpRequestType)
    Bases: object

class tmtccmd.cfdp.request.CfdpRequestWrapper(base: CfdpRequestBase | None)
    Bases: object

    property request: CfdpRequestType

    property request_type: CfdpRequestType

    to_put_request() → PutRequestCfgWrapper

class tmtccmd.cfdp.request.PutRequestCfgWrapper(cfg: CfdpParams)
    Bases: CfdpRequestBase
```

## 1.5.3 Application Package

### Module

Contains core methods called by entry point files to setup and start a tmtccmd application. It also re-exports commonly used classes and functions.

### Core Package

Contains core methods called by entry point files to setup and start a tmtccmd application. It also re-exports commonly used classes and functions.

```
tmtccmd.create_default_tmtc_backend(setup_wrapper: SetupWrapper, tm_handler: TmHandlerBase,
                                   tc_handler: TcHandlerBase, init_procedure: ProcedureWrapper |
                                   None) → BackendBase
```

Creates a default TMTC backend instance which can be passed to the tmtccmd runner

#### Parameters

- **init\_procedure**
- **tc\_handler**
- **setup\_wrapper**
- **tm\_handler**

#### Returns

```
tmtccmd.get_lib_logger() → Logger
```

Get the library logger. Please note that this logger can be configured by calling `init_logger()`. This might make sense depending on how the library logger is used.

```
tmtccmd.init_logger(propagate: bool = False, log_level: int = 20)
```

Initiate the library internal logger. There are various ways how to use the logging support of tmtccmd in an application.

1. Usage of a custom application logger, possibly a root logger created with `logging.getLogger()`. In that case, this function does not have to be called if the goal is to have the library logs be emitted by the custom logger. It might still make sense to apply the library console logging format to the application logger using `tmtccmd.logging.add_colorlog_console_logger()`.
2. Usage of a custom application logger, but the library logs should not be emitted by that logger. In that case, the propagation can be disabled but this function can be used to still set up the library logger.
3. No usage of logging in the application but the logs of the library should still be emitted. In that case, this function should still be called to set the log level and set up formatting. The propagation flag does not matter and can be left at the default level.

#### Parameters

- **propagate** – Specifies whether logs are propagated. If the user wants to use an own (root) logger and does not wish the logs of the library to be propagated to that logger, this should be set to `False`, which is the default.
- **log\_level** – Sets the log level of the library logger

`tmtccmd.init_printout`(*use\_gui*: *bool*)

`tmtccmd.setup`(*setup\_args*: *SetupWrapper*)

This function needs to be called first before running the TMTCC commander core. The setup arguments encapsulate all required arguments for the TMTCC commander.

#### Parameters

**setup\_args** – Setup arguments

`tmtccmd.setup_backend_def_procedure`(*backend*: *CcsdsTmtcBackend*, *tmtc\_params*: *TreeCommandingProcedure*)

`tmtccmd.start`(*tmtc\_backend*: *BackendBase*, *hook\_obj*: *HookBase*, *tmtc\_frontend*: *FrontendBase* | *None* = *None*, *app\_name*: *str* = 'TMTCC Commander')

This is the primary function to run the TMTCC commander. Users should call this function to start the TMTCC commander. Please note that `setup()` needs to be called before this function. You also need to build a TMTCC backend instance and pass it to this call. You can use `create_default_tmtc_backend()` to create a generic backend.

#### Parameters

- **tmtc\_backend** – Custom backend can be passed here. Otherwise, a default backend will be created
- **hook\_obj**
- **tmtc\_frontend** – Custom frontend can be passed here. Otherwise, a default frontend will be created
- **app\_name** – Name of application. Will be displayed in GUI

#### Raises

**RuntimeError** – if `setup()` was not called before

#### Returns

## Core Package

### Module contents

### Submodules

#### tmtccmd.core.ccsds\_backend module

```
class tmtccmd.core.ccsds_backend.CcsdsTmtcBackend(tc_mode: TcMode, tm_mode: TmMode, com_if: ComInterface, tm_listener: CcsdsTmListener, tc_handler: TcHandlerBase)
```

Bases: BackendBase

This is the primary class which handles TMTC reception and sending

**close\_com\_if()**

Closes the TM listener and the communication interface :return:

**property com\_if:** *ComInterface*

**com\_if\_active()**

**property com\_if\_id**

**property current\_procedure:** *ProcedureWrapper*

**default\_operation()**

Command handling. This is a convenience function to call the TM and the TC operation and then auto-determine the internal mode with the *mode\_to\_req()* method.

**Raises**

*NoValidProcedureSet* – No valid procedure set to be passed to the feed callback of the TC handler

**property inter\_cmd\_delay**

**mode\_to\_req()**

This function will convert the internal state of the backend to a backend *request*, which can be used to determine the next operation. These requests can be treated like recommendations. For example, for if both the TC and the TM mode are IDLE, the request will be set to BackendRequest.DELAY\_IDLE field.

**open\_com\_if()**

Start the backend. Raise RuntimeError on failure

**periodic\_op**(\_args: *any* | *None* = *None*) → BackendState

Periodic operation. Simply calls the *default\_operation()* function. :raises KeyboardInterrupt: Yields info output and then propagates the exception :raises IOError: Yields informative output and propagates exception :

**poll\_tm()**

Poll TM, irrespective of current TM mode

**register\_keyboard\_interrupt\_handler()**

Register a keyboard interrupt handler which closes the COM interface and prints a small message

**property request**

**start()**

**property state**

**property tc\_mode**

**tc\_operation()**

This function will handle consuming the current TC queue if one is available, or attempting to fetch a new one if it is not. This function will only do something if the *tc\_mode* is set to a non IDLE value.

It is necessary to set a valid procedure before calling this by using the *current\_proc\_info* setter function.

**Raises**

*NoValidProcedureSet* – No valid procedure set to be passed to the feed callback of the TC handler

**property tm\_listener**

**property tm\_mode**

**tm\_operation()**

This function will fetch and forward TM data from the current communication interface to the user TM handler. It only does so if the *tm\_mode* is set to the LISTENER mode

**try\_set\_com\_if**(*com\_if*: *ComInterface*) → bool

**exception** tmtccmd.core.ccsds\_backend.NoValidProcedureSet

Bases: *Exception*

## tmtccmd.core.base module

**class** tmtccmd.core.base.BackendRequest(*value*)

Bases: *IntEnum*

These requests can be treated like recommendations on what to do after calling the backend handler functions and the *BackendState.mode\_to\_req()* function.

**Brief explanation of fields:**

1. NONE: No special recommendation
2. TERMINATION\_NO\_ERROR: Will be returned for the One Queue mode after finishing queue handling.
3. DELAY\_IDLE: TC and TM mode are idle, so there is nothing to do
4. DELAY\_LISTENER: TC handling is not active but TM listening is active. Delay to wait for new TM packets
5. CALL\_NEXT: It is recommended to call the handler functions immediately, for example to handle the next entry in the TC queue

**CALL\_NEXT** = 5

**DELAY\_CUSTOM** = 4

**DELAY\_IDLE** = 2

**DELAY\_LISTENER** = 3

```
NONE = 0

TERMINATION_NO_ERROR = 1

class tmtccmd.core.base.FrontendBase
    Bases: object

    abstract start(args: Any)
        Start the frontend. :return:

class tmtccmd.core.base.ModeWrapper
    Bases: object

class tmtccmd.core.base.TcMode(value)
    Bases: IntEnum

    An enumeration.

    IDLE = 0

    MULTI_QUEUE = 2

    ONE_QUEUE = 1

class tmtccmd.core.base.TmMode(value)
    Bases: IntEnum

    An enumeration.

    IDLE = 0

    LISTENER = 1
```

### tmtccmd.core.globals\_manager module

```
tmtccmd.core.globals_manager.get_global(global_param_id: int, lock: bool = False)

tmtccmd.core.globals_manager.lock_global_pool(blocking: bool | None = None, timeout: float | None =
                                                None) → bool

    Lock the global objects. This is important if the values are changed. Don't forget to unlock the pool after finishing
    work with the globals! :param timeout_seconds: Attempt to lock for this many second. Default value -1 blocks
    permanently until lock is released. :return: Returns whether lock was locked or not.

tmtccmd.core.globals_manager.set_lock_timeout(timeout: float)

tmtccmd.core.globals_manager.unlock_global_pool()

tmtccmd.core.globals_manager.update_global(global_param_id: int, parameter: any, lock: bool = False)
```

## Telemetry Handling Package

## Telemetry Handling Package

### CCSDS TM Listener Module

Contains the TmListener which can be used to listen to Telemetry in the background

**class** tmtccmd.tmtc.ccsds\_tm\_listener.CcsdsTmListener(tm\_handler: CcsdsTmHandler)

Bases: `object`

Simple helper object which can be used for retrieving and routing CCSDS packets. It can be used to poll CCSDS packets from a provided tmtccmd.com\_if.ComInterface and then route them using a provided CCSDS TM handler.

**operation**(com\_if: ComInterface) → int

Core operation to route packet to the provided handler.

#### Parameters

**com\_if**

#### Raises

**PacketsTooSmallForCcsds** – If any of the received packets are too small. The internal handler will still continue to process the remaining packet list retrieved from the COM interface.

#### Returns

**exception** tmtccmd.tmtc.ccsds\_tm\_listener.PacketsTooSmallForCcsds(packets: List[bytes])

Bases: `Exception`

### TM Common Module

**class** tmtccmd.tmtc.common.CcsdsTmHandler(generic\_handler: GenericApidHandlerBase | None)

Bases: `TmHandlerBase`

Generic CCSDS handler class. The user can create an instance of this class to handle CCSDS packets by adding dedicated APID handlers or a generic handler for all APIDs with no dedicated handler

**add\_apid\_handler**(handler: SpecificApidHandlerBase)

Add a TM handler for a certain APID. The handler is a callback function which will be called if telemetry with that APID arrives.

#### Parameters

**handler** – Handler class instance

#### Returns

**handle\_packet**(apid: int, packet: bytes) → bool

Handle a packet with an APID. If a handler exists for the given APID, it is used to handle the packet. If not, a dedicated handler for unknown APIDs is called.

#### Parameters

- **apid**
- **packet**

**Returns**

True if the packet was passed to as dedicated APID handler, False otherwise

**has\_apid**(apid: *int*) → bool

**user\_hook**(apid: *int*, packet: *bytes*)

Can be overridden to trace all packets received.

**class** tmtccmd.tmtc.common.DefaultApidHandler(*user\_args: Any*)

Bases: [GenericApidHandlerBase](#)

**handle\_tm**(apid: *int*, \_packet: *bytes*, \_user\_args: *Any*)

**class** tmtccmd.tmtc.common.GenericApidHandlerBase(*user\_args: Any*)

Bases: [ABC](#)

This class is similar to the [SpecificApidHandlerBase](#) but it is not specific for an APID and the found APID will be passed to the callback

**abstract handle\_tm**(apid: *int*, \_packet: *bytes*, \_user\_args: *Any*)

**class** tmtccmd.tmtc.common.SpecificApidHandlerBase(*apid: int, user\_args: Any*)

Bases: [ABC](#)

Abstract base class for an CCSDS APID specific handler. The user can implement a TM handler by implementing this class and then adding it to the [CcsdsTmHandler](#). If a CCSDS space packet with a specific APID is received, it will be routed to this handler using the [handle\\_tm\(\)](#) callback function

**abstract handle\_tm**(\_packet: *bytes*, \_user\_args: *Any*)

**class** tmtccmd.tmtc.common.TmHandlerBase(*tm\_type: TmTypes*)

Bases: [object](#)

**get\_type**()

**class** tmtccmd.tmtc.common.TmTypes(*value*)

Bases: [Enum](#)

An enumeration.

CCSDS\_SPACE\_PACKETS = <class 'enum.auto'>

NONE = <class 'enum.auto'>

**TM Base Module (deprecated)**

**class** tmtccmd.tmtc.tm\_base.PusTmBase(*pus\_tm: PusTm*)

Bases: [PusTmInterface](#)

**property** apid: *int*

**pack**() → bytearray

**property** service: *int*

**property** ssc: *int*

**property** subservice: *int*



**property** `tm_data`: `bytes`

**class** `tmtccmd.tmtc.tm_base.PusTmInfoBase`(*pus\_tm*: *PusTm*)

Bases: *PusTmInfoInterface*

**append\_packet\_info**(*info*: *str*)

**append\_telemetry\_column\_headers**(*header\_list*: *list*)

Default implementation adds the PUS header content header (confusing, I know) to the list which can then be printed with a simple `print()` command. To add additional headers, override this method. Any child class should call this function as well if header information is required.

**Parameters**

**header\_list** – Header content will be appended to this list

**Returns**

**append\_telemetry\_content**(*content\_list*: *list*)

Default implementation adds the PUS header content to the list which can then be printed with a simple `print()` command. To add additional content, override this method. Any child class should call this function as well if header information is required.

**Parameters**

**content\_list** – Header content will be appended to this list

**Returns**

**get\_custom\_printout**() → *str*

**get\_print\_info**() → *str*

**get\_source\_data\_string**(*print\_format*: *PrintFormats* = *PrintFormats.HEX*) → *str*

**set\_custom\_printout**(*custom\_string*: *str*)

**set\_packet\_info**(*print\_info*: *str*)

**class** `tmtccmd.tmtc.tm_base.PusTmInfoInterface`

Bases: *object*

**abstract** **append\_packet\_info**(*info*: *str*)

**abstract** **append\_telemetry\_column\_headers**(*header\_list*: *list*)

**abstract** **append\_telemetry\_content**(*content\_list*: *list*)

**abstract** **get\_custom\_printout**() → *str*

**abstract** **get\_print\_info**() → *str*

**abstract** **get\_source\_data\_string**() → *str*

**abstract** **set\_packet\_info**(*print\_info*: *str*)

**class** `tmtccmd.tmtc.tm_base.PusTmInterface`

Bases: *object*

**abstract** **property** **apid**: *int*

**abstract** **pack**() → *bytearray*

```
abstract property service: int
abstract property ssc: int
abstract property subservice: int
abstract property tm_data: bytes
abstract property valid: bool
```

## Telecommand Handling Package

## Telecommand Handling Package

### TC Handler Submodule

```
class tmtccmd.tmtc.handler.FeedWrapper(queue_wrapper: QueueWrapper, auto_dispatch: bool)
```

Bases: `object`

This class wraps the queue and some additional information and useful fields which can be set by the user.

#### Variables

- **queue\_helper** – Can be used to simplify insertion of queue entries like telecommands into the queue
- **dispatch\_next\_queue** – Can be used to prevent the dispatch of the queue
- **modes** – Currently contains the current TC Mode and TM mode of the calling handler class

```
class tmtccmd.tmtc.handler.SendCbParams(info: ProcedureWrapper, entry: QueueEntryHelper, com_if: ComInterface)
```

Bases: `object`

Wrapper for all important parameters passed to the TC send callback.

#### Variables

- **info** – Procedure info about the procedure this queue entry is related too
- **entry** – Queue entry base type. The user can cast this back to the concrete type or just use duck typing if the concrete type is known
- **com\_if** – Communication interface. Will generally be used to send the packet, using the `tmtccmd.com_if.ComInterface.send()` method

```
class tmtccmd.tmtc.handler.TcHandlerBase
```

Bases: `ABC`

Generic abstract class for a TC handler object. Should be implemented by the user. This object then takes care of sending packets by providing the `send_cb()` send-callback. It also provides telecommand queues by providing the `feed_cb()` queue feeder callback.

```
abstract feed_cb(info: ProcedureWrapper, wrapper: FeedWrapper)
```

This function will be called to retrieve a telecommand queue from the user code, based on a procedure. The passed feed wrapper can be used to set the TC queue or other parameter like the inter-packet delay.

#### Parameters

- **info** – Generic base class for a procedure. For example, the `py:class:tmtccmd.tmtc.DefaultProcedureInfo` class uses a service string and op code string which can be used in the user code to select between different telecommand queues being packed
- **wrapper** – Wrapper type around the queue. It also contains a queue helper class to simplify adding entries to the telecommand queue

#### Returns

**abstract queue\_finished\_cb**(*info*: [ProcedureWrapper](#))

**abstract send\_cb**(*send\_params*: [SendCbParams](#))

This function callback will be called for each queue entry. This also includes miscellaneous queue entries, for example the ones used to log additional information. It is up to the user code implementation to determine the concrete queue entry and what to do with it.

In general, an implementation will perform the following steps:

1. Determine the queue entry and what to do with it
2. If applicable, retrieve the raw data to send from the queue entry and send it using the generic communication interface

All delay related entries will generally be handled by the send queue consumer so there is no need to manually delay the application in this callback. However, the queue consumer will not handle log entries so the user needs to take care of handling these entries and log the content to a console, file logger or any other system used to log something.

#### Parameters

**send\_params**

## TC Queue Submodule

```
class tmtccmd.tmtc.queue.DefaultPusQueueHelper(queue_wrapper: QueueWrapper,
                                              tc_sched_timestamp_len: int, seq_cnt_provider:
                                              ProvidesSeqCount | None, pus_verificator:
                                              PusVerificator | None, default_pus_apid: int | None)
```

Bases: [QueueHelperBase](#)

Default PUS Queue Helper which simplifies inserting PUS telecommands into the queue. It also provides a way to optionally stamp common PUS TC fields which would otherwise add boilerplate code during the packet creation process. This includes the following packet properties and it is also able to add the telecommand into a provided PUS verificator.

This queue helper also has special support for PUS 11 time tagged PUS telecommands and will perform its core functionality for the time-tagged telecommands as well.

**add\_ccsds\_tc**(*space\_packet*: [SpacePacket](#))

**add\_pus\_tc**(*pus\_tc*: [PusTc](#))

**pre\_add\_cb**(*entry*: [TcQueueEntryBase](#))

```
class tmtccmd.tmtc.queue.LogQueueEntry(log_str: str)
```

Bases: [TcQueueEntryBase](#)

```
class tmtccmd.tmtc.queue.PacketDelayEntry(delay_time: timedelta)
```

Bases: [TcQueueEntryBase](#)

```
    classmethod from_millis(millis: int) → PacketDelayEntry

class tmtccmd.tmtc.queue.PusTcEntry(pus_tc: PusTc)
    Bases: TcQueueEntryBase

class tmtccmd.tmtc.queue.QueueEntryHelper(base: TcQueueEntryBase | None)
    Bases: object
    property entry_type: TcQueueEntryType
    property is_tc: bool
    to_log_entry() → LogQueueEntry
    to_packet_delay_entry() → PacketDelayEntry
    to_pus_tc_entry() → PusTcEntry
    to_raw_tc_entry() → RawTcEntry
    to_space_packet_entry() → SpacePacketEntry
    to_wait_entry() → WaitEntry

class tmtccmd.tmtc.queue.QueueHelperBase(queue_wrapper: QueueWrapper)
    Bases: ABC
    add_log_cmd(print_str: str)
    add_packet_delay(delay: timedelta)
    add_packet_delay_ms(delay_ms: int)
    add_raw_tc(tc: bytes)
    add_wait(wait_time: timedelta)
    add_wait_ms(wait_ms: int)
    add_wait_seconds(wait_seconds: float)
    empty() → bool
    abstract pre_add_cb(entry: TcQueueEntryBase)

class tmtccmd.tmtc.queue.QueueWrapper(info: TcProcedureBase, queue: Deque[TcQueueEntryBase],
                                     inter_cmd_delay: timedelta = datetime.timedelta(0))
    Bases: object
    classmethod empty()

class tmtccmd.tmtc.queue.RawTcEntry(tc: bytes)
    Bases: TcQueueEntryBase

class tmtccmd.tmtc.queue.SpacePacketEntry(space_packet: SpacePacket)
    Bases: TcQueueEntryBase

class tmtccmd.tmtc.queue.TcQueueEntryBase(etype: TcQueueEntryType)
    Bases: object
    Generic TC queue entry abstraction. This allows filling the TC queue with custom objects
```

`is_tc()` → `bool`

Check whether concrete object is an actual telecommand

**class** `tmtccmd.tmtc.queue.TcQueueEntryType(value)`

Bases: `Enum`

An enumeration.

`CCSDS_TC` = `'ccsds-tc'`

`CUSTOM` = `'custom'`

`LOG` = `'log'`

`PACKET_DELAY` = `'set-delay'`

`PUS_TC` = `'pus-tc'`

`RAW_TC` = `'raw-tc'`

`WAIT` = `'wait'`

**class** `tmtccmd.tmtc.queue.WaitEntry(wait_time: timedelta)`

Bases: `TcQueueEntryBase`

**classmethod** `from_millis(millis: int)` → `WaitEntry`

## TC Procedure Submodule

**class** `tmtccmd.tmtc.procedure.CfdpProcedure`

Bases: `TcProcedureBase`

**property** `cfdp_request_type`

**class** `tmtccmd.tmtc.procedure.CustomProcedureInfo(procedure: Any)`

Bases: `TcProcedureBase`

**class** `tmtccmd.tmtc.procedure.ProcedureWrapper(procedure: TcProcedureBase | None)`

Bases: `object`

Procedure helper class. It wraps the concrete procedure object but allows easily casting it to concrete types supported by the framework.

**property** `proc_type`

**to\_cfdp\_procedure()** → `CfdpProcedure`

**to\_custom\_procedure()** → `CustomProcedureInfo`

**to\_tree\_commanding\_procedure()** → `TreeCommandingProcedure`

**class** `tmtccmd.tmtc.procedure.TcProcedureBase(p_type: TcProcedureType)`

Bases: `object`

**class** `tmtccmd.tmtc.procedure.TcProcedureType(value)`

Bases: `Enum`

An enumeration.

CFDP = 1

CUSTOM = 2

TREE\_COMMANDING = 0

**class** tmtccmd.tmtc.procedure.TreeCommandingProcedure(cmd\_path: str | None)

Bases: *TcProcedureBase*

Generic abstraction for procedures. A procedure can be a single command or a sequence of commands. Generally, one procedure is mapped to a specific TC queue which is packed during run-time

**classmethod** empty()

## Sequential CCSDS Sender Submodule

Used to send multiple TCs in sequence

**class** tmtccmd.tmtc.ccsds\_seq\_sender.SenderMode(value)

Bases: *IntEnum*

An enumeration.

BUSY = 0

DONE = 1

**class** tmtccmd.tmtc.ccsds\_seq\_sender.SeqResultWrapper(mode: SenderMode)

Bases: *object*

**class** tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender(queue\_wrapper: QueueWrapper, tc\_handler: TcHandlerBase)

Bases: *object*

Specific implementation of CommandSenderReceiver to send multiple telecommands in sequence

**handle\_new\_queue\_forced**(queue\_wrapper: QueueWrapper)

**handle\_non\_tc\_entry**(queue\_entry: TcQueueEntryBase) → bool

Checks whether the entry in the pus\_tc queue is a telecommand. :param queue\_entry: Generic queue entry  
:return: True if queue entry is telecommand, False if it is not

**property** mode

**no\_delay\_remaining**() → bool

**operation**(com\_if: ComInterface) → SeqResultWrapper

Primary function which should be called periodically to consume a TC queue.

### Parameters

**com\_if** – Communication interface used to send telecommands. Will be passed to the user send function

**property** queue\_wrapper

**resume**()

Can be used to resume a finished sequential sender if the provided queue is not empty anymore

## 1.5.4 Application Configuration Package

Configuration helpers and definitions.

Submodules:

- `tmtccmd.config.hook` - Base hook class which should be implemented by user and is used by the framework to retrieve certain configuration from the user.
- `tmtccmd.config.args` - Various helper methods and classes to create the argument parsers and arguments converts to create the data structures expected by this library from passed CLI arguments.

### Configuration Package

#### Module

Configuration helpers and definitions.

Submodules:

- `tmtccmd.config.hook` - Base hook class which should be implemented by user and is used by the framework to retrieve certain configuration from the user.
- `tmtccmd.config.args` - Various helper methods and classes to create the argument parsers and arguments converts to create the data structures expected by this library from passed CLI arguments.

```
class tmtccmd.config.SetupWrapper(hook_obj: HookBase, setup_params: SetupParams,
                                   proc_param_wrapper: ProcedureParamsWrapper, json_cfg_path: str |
                                   None = None)
```

Bases: `object`

This class encapsulates various important setup parameters required by tmtccmd components

**property params**

```
tmtccmd.config.backend_mode_conversion(mode: str, mode_wrapper: ModeWrapper)
```

```
tmtccmd.config.cfdp_put_req_params_to_procedure(params: CfdpParams) → CfdpProcedure
```

```
tmtccmd.config.params_to_procedure_conversion(param_wrapper: ProcedureParamsWrapper) →
                                             ProcedureWrapper
```

```
tmtccmd.config.tmtc_params_to_procedure(params: TreeCommandingParams) →
                                             TreeCommandingProcedure
```

### Configuration Hook Submodule

```
class tmtccmd.config.hook.HookBase(cfg_file_path: str | None = None)
```

Bases: `ABC`

This hook allows users to adapt the TMTC commander core to the unique mission requirements. It is used by implementing all abstract functions and then passing the instance to the TMTC commander core.

```
assign_communication_interface(com_if_key: str) → ComInterface | None
```

Assign the communication interface used by the TMTC commander to send and receive TMTC with.

**Parameters**

**com\_if\_key** – String key of the communication interface to be created.

Deprecated since version 8.0.0rc0: implement and use `get_communication_interface` instead

**get\_cmd\_history()** → `History` | `None`

Optionlly return a history class for the past command paths which will be used when prompting a command path from the user in CLI mode.

**get\_com\_if\_dict()** → `Mapping[str, Tuple[str, Any]]`

**abstract get\_command\_definitions()** → `CmdTreeNode`

This function should return the root node of the command definition tree.

**abstract get\_communication\_interface(*com\_if\_key: str*)** → `ComInterface` | `None`

**get\_object\_ids()** → `Mapping[bytes, ComponentIdBase]`

Deprecated since version 8.0.0: application specific code

**get\_tmtc\_definitions()** → `TmtcDefinitionWrapper`

This is a dicitonary mapping services represented by strings to an operation code dictionary.

#### Returns

Deprecated since version 8.0.0rc0: implement and use `get_command_definitions` instead

**perform\_mode\_operation(*tmtc\_backend: BackendBase, mode: int*)**

Perform custom mode operations.

#### Parameters

- **tmtc\_backend**
- **mode**

#### Returns

## Argument Parsing Submodule

Argument parser module.

```
class tmtccmd.config.args.AppParams(use_gui: 'bool' = False, reduced_printout: 'bool' = False,
                                    use_ansi_colors: 'bool' = True, compl_style: 'CompleteStyle' =
                                    <CompleteStyle.READLINE_LIKE: 'READLINE_LIKE'>)
```

Bases: `object`

**compl\_style:** `CompleteStyle` = `'READLINE_LIKE'`

**reduced\_printout:** `bool` = `False`

**use\_ansi\_colors:** `bool` = `True`

**use\_gui:** `bool` = `False`

```
class tmtccmd.config.args.BackendParams(mode: 'str' = "", com_if_id: 'str' = "", listener: 'bool' = False,
                                         interactive: 'bool' = False)
```

Bases: `object`

**com\_if\_id:** `str` = `''`

**interactive:** `bool` = `False`



```
listener: bool = False
```

```
mode: str = ''
```

```
class tmtccmd.config.args.CommandingParams(delay: float = 0.0, apid: int = 0, print_tree: bool = False,
                                           tree_print_with_description: bool = True,
                                           tree_print_max_depth: Optional[int] = None)
```

Bases: `object`

```
apid: int = 0
```

```
delay: float = 0.0
```

```
print_tree: bool = False
```

```
tree_print_max_depth: int | None = None
```

```
tree_print_with_description: bool = True
```

```
class tmtccmd.config.args.PostArgsParsingWrapper(args_raw: Namespace, unknown_args: List[str],
                                                  params: SetupParams, hook_obj: HookBase)
```

Bases: `object`

This helper class helps with the internalization of the parse arguments into the format expected by tmtccmd.

#### Variables

**assign\_com\_if** – If this is set to True (default), the wrapper will try to create a `tmtccmd.com.ComInterface` on the conversion methods.

**request\_type\_from\_args()** → `TcProcedureType`

**set\_cfdp\_params\_with\_prompts**(*cfdp\_params*: `CfdpParams`)

**set\_cfdp\_params\_without\_prompts**(*cfdp\_params*: `CfdpParams`)

**set\_params\_with\_prompts**(*proc\_base*: `ProcedureParamsWrapper`)

**set\_params\_without\_prompts**(*proc\_wrapper*: `ProcedureParamsWrapper`)

**set\_tmtc\_params\_with\_prompts**(*tmtc\_params*: `TreeCommandingParams`)

**set\_tmtc\_params\_without\_prompts**(*tmtc\_params*: `TreeCommandingParams`)

Set up the parameter object from the parsed arguments. This call auto-determines whether prompts should be used depending on whether the GUI flag was passed or not.

#### Raises

**Value Error** – Parse function call missing

**property use\_gui**

This only yields valid values if `parse()` was called once

```
class tmtccmd.config.args.PreArgsParsingWrapper
```

Bases: `object`

This class can be used to simplify parsing all tmtccmd CLI arguments.

It wraps a parent parser and an argument parser but is also able to create default parsers. The `parse()` method can be used to convert this parse the CLI arguments and then create a `PostArgsParsingWrapper` to process these arguments.

Please note that the parent parser and argument parser field have to be set or created first after creating this wrapper. They can be created by calling `create_default_parent_parser()` and the `create_default_parser()`, but the second function requires the parent parser to be set or created first.

**add\_cfdp\_args()**

Add the default CFDP procedure parameters to the default parser.

**add\_def\_proc\_and\_cfdp\_as\_subparsers()** → `Tuple[ArgumentParser, ArgumentParser]`

Add the default tmtc and cfdp procedure as subparsers.

**add\_def\_proc\_args()**

Add the default tmtc procedure parameters to the default parser. This includes the service and operation code flags.

**create\_default\_parent\_parser()**

Create a default parent parser, which contains common flags for all possible tmtccmd submodules. For example, both the cfdp and default tmtc submodule could contain these common flags. The user can extend or modify the parent parser after it was created.

**create\_default\_parser()**

Create the default parser. Requires a valid parent parser containing common flags, The user can create or modify the parser after it was created. This function requires a valid parent parser to be set or created via `create_default_parent_parser()`.

**parse()**(hook\_obj: `HookBase`, setup\_params: `SetupParams`) → `PostArgsParsingWrapper`

Parses the set parser by calling the `argparse.ArgumentParser.parse_known_args()` method internally and returns the `PostArgsParsingWrapper` to simplify processing the arguments.

**class** tmtccmd.config.args.ProcedureParamsWrapper

Bases: `object`

**cfdp\_params()** → `CfdpParams` | `None`

**property** ptype

**set\_params**(params: `TreeCommandingParams` | `CfdpParams`)

**tree\_commanding\_params()** → `TreeCommandingParams` | `None`

**class** tmtccmd.config.args.SetupParams(*com\_if*: `ComInterface` | `None` = `None`, *cmd\_params*: `CommandingParams` | `None` = `None`, *backend\_params*: `BackendParams` | `None` = `None`, *app\_params*: `AppParams` | `None` = `None`)

Bases: `object`

**property** apid

**property** com\_if\_id

**property** mode

**property** use\_gui

tmtccmd.config.args.add\_cfdp\_procedure\_arguments(parser\_or\_subparser: `ArgumentParser`)

TODO: Could be extended to support the various types of CFDP user primitives. Right now, the first thing to be implemented will be the put request

tmtccmd.config.args.add\_default\_com\_if\_arguments(arg\_parser: `ArgumentParser`)

```

tmtccmd.config.args.add_default_tmtccmd_args(parser: ArgumentParser)
tmtccmd.config.args.add_ethernet_arguments(arg_parser: ArgumentParser)
tmtccmd.config.args.add_generic_arguments(arg_parser: ArgumentParser)
tmtccmd.config.args.add_tmtc_listener_arg(arg_parser: ArgumentParser)
tmtccmd.config.args.add_tmtc_mode_arguments(arg_parser: ArgumentParser)
tmtccmd.config.args.add_tree_commanding_arguments(parser_or_subparser: ArgumentParser)
tmtccmd.config.args.args_to_all_params_for_cfdp(pargs: Namespace, params: SetupParams,
                                                cfdp_params: CfdpParams, hook_obj: HookBase,
                                                use_prompts: bool, assign_com_if: bool)

```

Helper function to convert CFDP command line arguments to the setup parameters.

```

tmtccmd.config.args.args_to_all_params_tmtc(pargs: Namespace, params: SetupParams,
                                             def_tmtc_params: TreeCommandingParams, hook_obj:
                                             HookBase, use_prompts: bool, assign_com_if: bool =
                                             True)

```

This function converts command line arguments to the internalized setup parameters.

It is recommended to use the PostArgsParsingHelper class to do this instead of calling this function directly.

If some arguments are unspecified, they are set here with (variable) default values.

#### Parameters

- **pargs** – Parsed arguments from calling parse method.
- **params** – Setup parameter object which will be set by this function.
- **hook\_obj**
- **def\_tmtc\_params**
- **use\_prompts** – Specify whether terminal prompts are allowed to retrieve unspecified arguments. For something like a GUI, it might make sense to disable this
- **assign\_com\_if** – Specifies whether this function should try to determine the COM interface from the specified key.

#### Returns

None

```

tmtccmd.config.args.args_to_params_generic(pargs: Namespace, params: SetupParams, hook_obj:
                                           HookBase, use_prompts: bool, assign_com_if: bool)

tmtccmd.config.args.cfdp_args_to_cfdp_params(pargs: Namespace, cfdp_params: CfdpParams)
    Convert the argument parser CFDP arguments provided by this library to the internalized tmtccmd.config.
    defs.CfdpParams dataclass.

tmtccmd.config.args.create_default_args_parser(parent_parser: ArgumentParser, descript_txt: str |
                                              None = None) → ArgumentParser

tmtccmd.config.args.determine_cmd_path(params: SetupParams, def_params: TreeCommandingParams,
                                       hook_obj: HookBase, pargs: Namespace, use_prompts: bool)

tmtccmd.config.args.get_default_descript_txt() → str

```

```
tmtccmd.config.args.parse_default_tmtccmd_input_arguments(args: Sequence[str], parser:
ArgumentParser, print_known_args:
bool = False, print_unknown_args: bool
= False) → Tuple[Namespace, List[str]]
```

Parses all input arguments by calling `argparse.ArgumentParser.parse_known_args()`. It is recommended to use the [PreArgsParsingWrapper](#) instead of using this function directly.

#### Parameters

- **args** – The actual full list of parse CLI arguments
- **parser** – The parser to be used.
- **print\_known\_args** – Debugging function to print all known arguments.
- **print\_unknown\_args**

#### Returns

Input arguments contained in a special namespace and accessible by `args.<variable>`

```
tmtccmd.config.args.perform_tree_printout(cmd_params: CommandingParams, cmd_def_tree:
CmdTreeNode)
```

## TMTC Configuration Submodule

```
class tmtccmd.config.tmtc.CmdTreeNode(name: str, description: str, parent: CmdTreeNode | None = None,
hide_children_for_print: bool = False,
hide_children_which_are_leaves: bool = False)
```

Bases: `object`

The command tree node is the primary data structure used to specify the command structure in a way it can be used by framework components.

The node class provides an API which allows to build a tree of command nodes. Generally, a full path from the root node to a leaf will be the command identifier or command path for executing a certain command or procedure.

You can create the root node using the `CmdTreeNode.root_node()` class method. After that children can be appended to the nodes using the `CmdTreeNode.add_child()` method.

You can use square bracket operator to access the children of a node directly. For example, if a node with the name `test_node` has the child `event`, you could use `test_node["event"]` to access the child node.

**add\_child**(child: `CmdTreeNode`)

Add a child to the node. This will also assign the parent class of the child to the current node.

**contains\_path**(path: `str`) → `bool`

Check whether a full slash separated command path is contained within the command tree.

**contains\_path\_from\_node\_list**(node\_name\_list: `List[str]`) → `bool`

Check whether the given list of nodes are contained within the command tree.

**extract\_subnode**(path: `str`) → `CmdTreeNode` | `None`

Extract a subnode given a relative path.

**extract\_subnode\_by\_node\_list**(node\_list: `List[str]`) → `CmdTreeNode` | `None`

Extract a subnode given a list which would form a relative path if it were joined using slashes.

**is\_leaf()** → bool

A leaf is a node which has no children.

**property name\_dict:** Dict[str, Dict[str, Any] | None]

Returns a nested dictionary where the key is always the name of the node, and the value is one nested name dictionary for each child node.

**classmethod root\_node()** → CmdTreeNode

**str\_for\_tree**(with\_description: bool, max\_depth: int | None = None, show\_hidden\_elements: bool = False) → str

Retrieve the a human readable printout of the tree.

#### Parameters

- **with\_description** – Display descriptions right to the tree.
- **max\_depth** – Entries will be cut-off at the specified depth. None can be specified to print all depths.
- **show\_hidden\_elements** – Overrides the hide argument of command tree nodes.

**class** tmtccmd.config.tmtc.DepthInfo(depth: int, last\_child: bool, max\_depth: int | None = None, layer\_is\_last\_set: Set[int] | None = None)

Bases: object

**clear\_layer\_is\_last\_child**(depth: int)

**is\_layer\_for\_last\_child**(depth: int) → bool

**set\_layer\_is\_last\_child**(depth: int)

**class** tmtccmd.config.tmtc.OpCodeEntry(\*args, \*\*kwargs)

Bases: object

Deprecated since version 8.0.0: use the new command definition tree instead

**add**(keys: str | ~typing.List[str], info: str, options: ~tmtccmd.config.tmtc.OpCodeOptionBase = <tmtccmd.config.tmtc.OpCodeOptionBase object>)

**info**(op\_code: str) → str | None

**property op\_code\_dict\_num\_keys**

**property op\_code\_dict\_str\_keys**

**sort\_num\_key\_dict**()

**sort\_text\_key\_dict**()

**class** tmtccmd.config.tmtc.OpCodeOptionBase(\*args, \*\*kwargs)

Bases: object

Deprecated since version 8.0.0: use the new command definition tree instead

**class** tmtccmd.config.tmtc.TmtcDefinitionWrapper(\*args, \*\*kwargs)

Bases: object

Deprecated since version 8.0.0: use the new command definition tree instead

**add\_service**(name: str, info: str, op\_code\_entry: OpCodeEntry)

`op_code_entry(service_name: str) → OpCodeEntry | None`

`sort()`

`class tmtccmd.config.tmtc.TreePart(value)`

Bases: `Enum`

An enumeration.

`BLANK = ' '`

`CORNER = '└─'`

`EDGE = '├─'`

`LINE = '│'`

`tmtccmd.config.tmtc.call_all_definitions_providers(defs: TmtcDefinitionWrapper, *args, **kwargs)`

Deprecated since version 8.0.0: use the new command definition tree instead

`tmtccmd.config.tmtc.tmtc_definitions_provider(adder_func)`

Function decorator which registers the decorated function to be a TMTC definition provider. The `execute_tmtc_def_functions()` function can be used to call all functions.

It is expected that the decorated function takes the `py:class:TmtcDefWrapper` as the first argument. The user can pass any additional arguments as positional and keyword arguments.

Deprecated since version 8.0.0: use the new command definition tree instead

## CFDP Configuration Submodule

`tmtccmd.config.cfdp.cfdp_req_to_put_req_get_req(params: CfdpParams, local_id: UnsignedByteField, remote_id: UnsignedByteField) → PutRequest | None`

This function converts the internalized CFDP parameters to the get request variant of the `tmtccmd.cfdp.request.PutRequest` class. Please note that the local ID refers to the receiver of the target of the file copy operation for a get request while the remote ID refers to the sender component for the file copy operation.

`tmtccmd.config.cfdp.cfdp_req_to_put_req_proxy_put_req(params: CfdpParams, dest_id_put_request: UnsignedByteField, dest_id_proxy_put_req: UnsignedByteField) → PutRequest | None`

Generic function to convert the internalized CFDP parameters to a proxy put request.

### Parameters

- **params** – CFDP parameters
- **dest\_id\_put\_request** – Recipient of the put request.
- **dest\_id\_proxy\_put\_req** – Recipient of the proxy put operation. For a get request, this should be the ID of the sender.

`tmtccmd.config.cfdp.cfdp_req_to_put_req_regular(params: CfdpParams, dest_id: UnsignedByteField) → PutRequest | None`

`tmtccmd.config.cfdp.generic_cfdp_params_to_put_request(params: CfdpParams, local_id: UnsignedByteField, remote_id: UnsignedByteField, dest_id_proxy_put_req: UnsignedByteField) → PutRequest | None`

Please note that this function currently only has the following functionality. It might be extended in the future to have more functionality, or be converted to a factory class.

1. Create a regular put request for a file copy operation.
2. Create a proxy put request.

## Communication Configuration Submodule

```
class tmtccmd.config.com.ComCfgBase(com_if_key: str, json_cfg_path: str)
```

Bases: `object`

```
class tmtccmd.config.com.SerialCfgWrapper(com_if_key: str, json_cfg_path: str, serial_cfg: SerialCfg)
```

Bases: `ComCfgBase`

```
class tmtccmd.config.com.TcpipCfg(if_type: TcpIpType, com_if_key: str, json_cfg_path: str, send_addr: EthAddr, space_packet_ids: Sequence[PacketId] | None, recv_addr: EthAddr | None = None)
```

Bases: `ComCfgBase`

```
tmtccmd.config.com.create_com_interface_cfg_default(com_if_key: str, json_cfg_path: str, space_packet_ids: Sequence[PacketId] | None)  
→ ComCfgBase | None
```

```
tmtccmd.config.com.create_com_interface_default(cfg: ComCfgBase) → ComInterface | None
```

Return the desired communication interface object

### Parameters

**cfg** – Generic configuration

### Returns

```
tmtccmd.config.com.create_default_serial_interface(com_if_key: str, json_cfg_path: str, serial_cfg: SerialCfg) → ComInterface | None
```

Create a default serial interface. Requires a certain set of global variables set up. See `set_up_serial_cfg()` for more details.

### Parameters

- **com\_if\_key**
- **json\_cfg\_path**
- **serial\_cfg** – Generic serial configuration parameters

### Returns

```
tmtccmd.config.com.create_default_tcpip_interface(tcpip_cfg: TcpipCfg) → ComInterface | None
```

Create a default serial interface. Requires a certain set of global variables set up. See `default_tcpip_cfg_setup()` for more details.

### Parameters

**tcpip\_cfg** – Configuration parameters

### Returns

tmtccmd.config.com.default\_serial\_cfg\_baud\_and\_port\_setup(json\_cfg\_path: str, cfg: SerialCfg)

Default setup for serial interfaces.

#### Parameters

- **json\_cfg\_path**
- **cfg** – The baud and serial port parameter will be set in this dataclass

#### Returns

tmtccmd.config.com.default\_tcpip\_cfg\_setup(com\_if\_key: str, tcpip\_type: TcpIpType, json\_cfg\_path: str, space\_packet\_ids: Sequence[PacketId] | None) → TcpipCfg

Default setup for TCP/IP communication interfaces. This instantiates all required data in the globals manager so a TCP/IP communication interface can be built with [create\\_default\\_tcpip\\_interface\(\)](#)

#### Parameters

- **com\_if\_key**
- **tcpip\_type**
- **json\_cfg\_path**
- **space\_packet\_ids** – Required if the TCP com interface needs to parse space packets

#### Returns

## Configuration Definitions Submodule

```
class tmtccmd.config.defs.CfdpParams(source_file: str = "", dest_file: str = "", closure_requested: bool = False, transmission_mode: TransmissionMode = TransmissionMode.UNACKNOWLEDGED, proxy_op: bool = False)
```

Bases: [object](#)

Simplified dataclass to model the most important CFDP parameters. This can for example be used to internalize CFDP CLI or GUI options.

**closure\_requested:** [bool](#) = False

**dest\_file:** [str](#) = ''

**proxy\_op:** [bool](#) = False

**source\_file:** [str](#) = ''

**transmission\_mode:** [TransmissionMode](#) = 1

```
class tmtccmd.config.defs.CoreComInterfaces(value)
```

Bases: [str](#), [Enum](#)

An enumeration.

**DUMMY** = 'dummy'

**SERIAL\_COBS** = 'serial\_cobs'

**SERIAL\_DLE** = 'serial\_dle'



```
SERIAL_QEMU = 'serial_qemu'
```

```
TCP = 'tcp'
```

```
UDP = 'udp'
```

```
UNSPECIFIED = 'unspec'
```

```
class tmtccmd.config.defs.CoreModeConverter
```

```
Bases: object
```

```
static get_str(mode: CoreModeList | int) → str
```

```
class tmtccmd.config.defs.CoreModeList(value)
```

```
Bases: IntEnum
```

These are the core modes which will be translated to different TC and TM modes for the CCSDS backend

1. ONE\_QUEUE\_MODE: This mode is optimized to handle one queue. It will configure the backend to request program termination upon finishing the queue handling. This is also the appropriate solution for single commands where the queue only consists of one telecommand.
2. LISTENER\_MODE: Only listen to TM
3. MULTI\_INTERACTIVE\_QUEUE\_MODE:

```
IDLE = 5
```

```
LISTENER_MODE = 1
```

```
MULTI_INTERACTIVE_QUEUE_MODE = 3
```

```
ONE_QUEUE_MODE = 0
```

```
class tmtccmd.config.defs.TreeCommandingParams(cmd_path: str | None)
```

```
Bases: object
```

```
cmd_path: str | None
```

```
tmtccmd.config.defs.default_json_path() → str
```

```
tmtccmd.config.defs.default_toml_path() → str
```

## Objects Submodule

```
tmtccmd.config.objects.get_base_component_id_mapping() → Mapping[bytes, ComponentIdBase]
```

These are the object IDs for the tmtccmd core. The core will usually take care of inserting these into the object manager during the program initialization.

:return Dictionary of the core object IDs

```
tmtccmd.config.objects.get_core_object_ids() → Mapping[bytes, ComponentIdBase]
```

## 1.5.5 Miscellaneous Modules

### Logging Module

### Logging Package

### Submodules

#### tmtccmd.logging.pus

```
class tmtccmd.logging.pus.RawTmtcLogBase(logger: Logger, log_repr: bool = True, log_raw_repr: bool = True)
```

Bases: *object*

**log\_bytes\_readable**(prefix: *str*, packet: *bytes*)

**log\_bytes\_repr**(prefix: *str*, packet: *bytes*)

**log\_repr**(prefix: *str*, packet: *PusTc* | *PusTm*)

**log\_tc**(packet: *PusTc*)

Default log function which logs the Python packet representation and raw bytes

**log\_tm**(packet: *PusTm*)

Default log function which logs the Python packet representation and raw bytes

**static tc\_prefix**(packet: *PusTc*, counter: *int*)

**static tm\_prefix**(packet: *PusTm*, counter: *int*)

```
class tmtccmd.logging.pus.RawTmtcRotatingLogWrapper(max_bytes: int, backup_count: int, logger: Logger | None = None, file_name: Path = PosixPath('log/tmtccmd_raw_pus'), suffix: str | None = '2024-05-16.log')
```

Bases: *RawTmtcLogBase*

```
class tmtccmd.logging.pus.RawTmtcTimedLogWrapper(when: TimedLogWhen, interval: int, logger: Logger | None = None, file_name: Path = PosixPath('log/tmtccmd_raw_pus'), suffix: str | None = '2024-05-16.log')
```

Bases: *RawTmtcLogBase*

```
class tmtccmd.logging.pus.RegularTmtcLogWrapper(file_name: Path | None = None, logger: Logger | None = None)
```

Bases: *object*

**classmethod get\_current\_tmtc\_file\_name**() → *Path*

```
class tmtccmd.logging.pus.TimedLogWhen(value)
```

Bases: *Enum*

An enumeration.

**PER\_DAY** = 'D'

**PER\_HOUR** = 'h'

```
PER_MINUTE = 'M'
```

```
PER_SECOND = 'S'
```

```
tmtccmd.logging.pus.date_suffix() → str
```

## Version Module

### Version Module

```
tmtccmd.version.get_version() → str
```

Retrieve the package version using the `importlib.metadata` API.

## Other Modules

### Utility Package

### Submodules

### JSON Module

```
class tmtccmd.util.json.JsonKeyNames(value)
```

Bases: `Enum`

An enumeration.

```
COM_IF = 'com_if'
```

```
SERIAL_BAUDRATE = 'serial_baudrate'
```

```
SERIAL_HINT = 'serial_hint'
```

```
SERIAL_PORT = 'serial_port'
```

```
TCPIP_TCP_DEST_IP_ADDRESS = 'tcpip_tcp_ip_addr'
```

```
TCPIP_TCP_DEST_PORT = 'tcpip_tcp_port'
```

```
TCPIP_TCP_RECV_MAX_SIZE = 'tcpip_tcp_recv_max_size'
```

```
TCPIP_UDP_DEST_IP_ADDRESS = 'tcpip_udp_ip_addr'
```

```
TCPIP_UDP_DEST_PORT = 'tcpip_udp_port'
```

```
TCPIP_UDP_RECV_IP_ADDRESS = 'tcpip_udp_recv_addr'
```

```
TCPIP_UDP_RECV_MAX_SIZE = 'tcpip_udp_recv_max_size'
```

```
TCPIP_UDP_RECV_PORT = 'tcpip_udp_recv_port'
```

```
tmtccmd.util.json.check_json_file(json_cfg_path: str) → bool
```

The check JSON file and return whether it was valid or not. A JSON file is invalid if it does not exist or the format is invalid. :return: True if JSON file is valid, False if not and a new one was created at the specified path

```
tmtccmd.util.json.save_to_json_with_prompt(key: str, value: Any, name: str, json_cfg_path: str, json_obj: Any) → bool
```

## Object ID Module

**class** tmtccmd.util.obj\_id.**ComponentIdBase**(obj\_id: int, byte\_len: int, name: str | None = None)

Bases: [UnsignedByteField](#)

Base class for unsigned object IDs with different byte sizes

**property** as\_hex\_string: str

**property** obj\_id: int

**class** tmtccmd.util.obj\_id.**ComponentIdU16**(obj\_id: int, name: str | None = None)

Bases: [ComponentIdBase](#)

A helper object for a unique object identifier which has a raw unsigned 16-bit representation.

**classmethod** from\_bytes\_typed(obj\_id\_as\_bytes: bytes) → [ComponentIdU16](#)

**class** tmtccmd.util.obj\_id.**ComponentIdU32**(obj\_id: int, name: str | None = None)

Bases: [ComponentIdBase](#)

A helper object for a unique object identifier which has a raw unsigned 32-bit representation.

```
>>> obj_id = ComponentIdU32(42, "Object with the answer to everything")
>>> int(obj_id)
42
>>> obj_id.name
'Object with the answer to everything'
>>> obj_id.as_bytes.hex(sep=",")
'00,00,00,2a'
>>> obj_id.as_hex_string
'0x0000002a'
```

**classmethod** from\_bytes\_typed(obj\_id\_as\_bytes: bytes) → [ComponentIdU32](#)

**class** tmtccmd.util.obj\_id.**ComponentIdU8**(obj\_id: int, name: str | None = None)

Bases: [ComponentIdBase](#)

A helper object for a unique object identifier which has a raw unsigned 8-bit representation.

**classmethod** from\_bytes\_typed(obj\_id\_as\_bytes: bytes) → [ComponentIdU8](#)

tmtccmd.util.obj\_id.**ObjectIdBase**

Deprecated type definition for [ComponentIdBase](#)

tmtccmd.util.obj\_id.**ObjectIdDictT**

Deprecated type definition for [ObjectIdMapping](#)

alias of [Mapping](#)[bytes, [ComponentIdBase](#)]

tmtccmd.util.obj\_id.**ObjectIdMapping**

Deprecated type definition for [ComponentIdMapping](#)

alias of [Mapping](#)[bytes, [ComponentIdBase](#)]

tmtccmd.util.obj\_id.**ObjectIdU16**

Deprecated type definition for [ComponentIdU16](#)

tmtccmd.util.obj\_id.ObjectIdU32

Deprecated type definition for *ComponentIdU32*

tmtccmd.util.obj\_id.ObjectIdU8

Deprecated type definition for *ComponentIdU8*

## Exit Module

class tmtccmd.util.exit.GracefulKiller

Bases: *object*

**exit\_gracefully()**

**kill\_now = False**

tmtccmd.util.exit.keyboard\_interrupt\_handler(*tmtc\_backend: BackendBase*)

## Hamming-Code Module

Hamming Code Implementation Hamming codes belong to the family of linear error correcting codes. Documentation: [https://en.wikipedia.org/wiki/Hamming\\_code](https://en.wikipedia.org/wiki/Hamming_code) They can be used to identify up to two bit errors and correct one bit error per 256 byte block.

class tmtccmd.util.hammingcode.HammingReturnCodes(*value*)

Bases: *Enum*

An enumeration.

**CODE\_OKAY = 0**

**ERROR\_ECC = 2**

**ERROR\_MULTI\_BIT = 3**

**ERROR\_SINGLE\_BIT = 1**

**OTHER\_ERROR = 4**

tmtccmd.util.hammingcode.hamming\_compute\_256(*data: bytearray*) → *bytearray*

Takes a bytearray with the size of 256 bytes and calculates the 22 parity bits for the hamming code which will be returned as a three byte bytearray.

### Parameters

**data**

### Returns

tmtccmd.util.hammingcode.hamming\_compute\_256x(*data: bytearray*) → *bytearray*

Computes 3-bytes hamming codes for a data block whose size is multiple of 256 bytes. Each 256 bytes block gets its own code.

### Parameters

**data** – Data to compute code for. Should be a multiple of 256 bytes, pad data with 0 if necessary!

### Returns

bytearray of hamming codes with the size (3 / 256 \* size). Empty bytearray if input is invalid.

```
tmtccmd.util.hammingcode.hamming_test()
```

Algorithm was verified with this simple test.

```
tmtccmd.util.hammingcode.hamming_verify_256(data: bytearray, original_hamming_code: bytearray) →  
HammingReturnCodes
```

Verifies and corrects a 256-bytes block of data using the given 22-bits hamming code. Returns 0 if there is no error, otherwise returns a HAMMING\_ERROR code.

#### Parameters

- **data** – 256 code block to verify
- **original\_hamming\_code** – Original 3 byte hamming code with 22 parity bits

#### Returns

See HammingReturnCodes enums. - -1 for invalid input - 0 if there are no errors. - 1 if there is a single bit error which has been corrected - 2 if the hamming code has been corrupted - 3 if there was a multi bit error which can not be corrected

```
tmtccmd.util.hammingcode.hamming_verify_256x(data: bytearray, original_hamming_code: bytearray) →  
HammingReturnCodes
```

## TMTC Printer (FSFW) Module

Contains classes and functions that perform all printing functionalities.

```
class tmtccmd.fsfw.tmtc_printer.DisplayMode(value)
```

Bases: `Enum`

List of display modes

```
LONG = <class 'enum.auto'>
```

```
SHORT = <class 'enum.auto'>
```

```
class tmtccmd.fsfw.tmtc_printer.FsfwTmTcPrinter(file_logger: Logger | None, display_mode:  
DisplayMode = DisplayMode.SHORT)
```

Bases: `object`

This class handles printing to the command line and to files

```
static bit_extractor(byte: int, position: int)
```

#### Parameters

- **byte**
- **position**

#### Returns

```
static chunks(lst: List, n) → Generator[List[List], None, None]
```

Yield successive n-sized chunks from lst.

```
generic_hk_tm_print(content_type: HkContentType, object_id: ComponentIdU32, set_id: int, hk_data:  
bytes)
```

This function pretty prints HK packets with a given header and content list :param content\_type: Type of content for HK packet :param object\_id: Object ID of the HK source :param set\_id: Unique set ID for the HK packet :param hk\_data: User defined HK data :return:

```
static generic_short_string(packet_if: PusTmInterface) → str
```

```
static get_validity_buffer_str(Validity_buffer: bytes, num_vars: int) → str
```

#### Parameters

- **validity\_buffer** – Validity buffer in bytes format
- **num\_vars** – Number of variables

#### Returns

```
handle_long_tm_print(packet_if: PusTmInterface, info_if: PusTmInfoInterface)
```

Main function to print the most important information inside the telemetry :param packet\_if: Core packet interface :param info\_if: Information interface :return:

```
static print_data(data: bytes)
```

#### Parameters

**data** – Data to print

#### Returns

None

```
print_validity_buffer(Validity_buffer: bytes, num_vars: int)
```

```
tmtccmd.fsfw.tmtc_printer.get_validity_buffer_str(Validity_buffer: bytes, num_vars: int) → str
```

#### Parameters

- **validity\_buffer** – Validity buffer in bytes format
- **num\_vars** – Number of variables

#### Returns

## Configuration Utility Module

```
class tmtccmd.util.conf_util.AnsiColors
```

Bases: `object`

```
BLUE = '\x1b[34m'
```

```
BOLD = '\x1b[1m'
```

```
CYAN = '\x1b[36m'
```

```
GREEN = '\x1b[32m'
```

```
MAGNETA = '\x1b[35m'
```

```
RED = '\x1b[31m'
```

```
RESET = '\x1b[0m'
```

```
YELLOW = '\x1b[33m'
```

```
tmtccmd.util.conf_util.acquire_timeout(lock, timeout)
```

Helper functions which allows to check result of the acquire operation while also using the context manager.  
:param lock: :param timeout: :return:

`tmtccmd.util.conf_util.check_args_in_dict`(*param*: any, *iterable*: *Iterable* | *dict*, *warning\_hint*: *str*) → *Tuple*[*bool*, *int*]

This functions checks whether the integer representation of a given parameter is contained within the passed collections, for example an (integer) enumeration. Please note that if the passed parameter has a string representation but is a digit, this function will attempt to check whether the integer representation is contained inside the passed enumeration. :param param: Value to be checked :param iterable: Enumeration, for example a `enum.Enum` or `enum.IntEnum` implementation :param warning\_hint: :return:

`tmtccmd.util.conf_util.wrapped_prompt`(*text*: *str*)

## Module contents

### Countdown Module

The countdown module was moved to the *spacepackets* library. Use the `spacepackets.countdown` module.

### Sequence Count Module

The sequence count module was moved to the *spacepackets* library. Use the `spacepackets.seqcount` module.

## Flight Software Framework (FSFW) Support Package

### Submodules

#### Module contents

`tmtccmd.fsfw.bit_extractor`(*byte*: *int*, *position*: *int*)

##### Parameters

- **byte**
- **position**

##### Returns

`tmtccmd.fsfw.parse_fsfw_events_csv`(*csv\_file*: *str*) → *Dict*[*int*, *EventInfo*] | *None*

`tmtccmd.fsfw.parse_fsfw_objects_csv`(*csv\_file*: *str*) → *Mapping*[*bytes*, *ComponentIdBase*] | *None*

`tmtccmd.fsfw.parse_fsfw_returnvalues_csv`(*csv\_file*: *str*) → *Mapping*[*int*, *RetValInfo*] | *None*

`tmtccmd.fsfw.validity_buffer_list`(*validity\_buffer*: *bytes*, *num\_vars*: *int*) → *List*[*bool*]

##### Parameters

- **validity\_buffer** – Validity buffer in bytes format
- **num\_vars** – Number of variables

##### Returns



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### t

- tmtccmd, 38
- tmtccmd.cfdp.request, 38
- tmtccmd.com, 10
- tmtccmd.com.dummy, 18
- tmtccmd.com.qemu, 20
- tmtccmd.com.ser\_utils, 18
- tmtccmd.com.serial\_base, 17
- tmtccmd.com.serial\_cobs, 15
- tmtccmd.com.serial\_dle, 16
- tmtccmd.com.tcp, 11
- tmtccmd.com.tcpip\_utils, 12
- tmtccmd.com.udp, 13
- tmtccmd.com.utils, 20
- tmtccmd.config, 51
- tmtccmd.config.args, 52
- tmtccmd.config.cfdp, 58
- tmtccmd.config.com, 59
- tmtccmd.config.defs, 60
- tmtccmd.config.hook, 51
- tmtccmd.config.objects, 61
- tmtccmd.config.tmtc, 56
- tmtccmd.core, 40
- tmtccmd.core.base, 41
- tmtccmd.core.ccsds\_backend, 40
- tmtccmd.core.globals\_manager, 42
- tmtccmd.fsfw, 68
- tmtccmd.fsfw.tmtc\_printer, 66
- tmtccmd.logging.pus, 62
- tmtccmd.pus, 24
- tmtccmd.pus.s11\_tc\_sched, 29
- tmtccmd.pus.s11\_tc\_sched\_defs, 29
- tmtccmd.pus.s17\_test, 30
- tmtccmd.pus.s17\_test\_defs, 30
- tmtccmd.pus.s1\_verification, 25
- tmtccmd.pus.s200\_fsfw\_mode, 36
- tmtccmd.pus.s200\_fsfw\_mode\_defs, 36
- tmtccmd.pus.s201\_fsfw\_health, 37
- tmtccmd.pus.s201\_fsfw\_health\_defs, 37
- tmtccmd.pus.s20\_fsfw\_param, 31
- tmtccmd.pus.s20\_fsfw\_param\_defs, 31
- tmtccmd.pus.s5\_fsfw\_event, 28
- tmtccmd.pus.s5\_fsfw\_event\_defs, 28
- tmtccmd.pus.s5\_satrs\_event, 27
- tmtccmd.pus.s5\_satrs\_event\_defs, 27
- tmtccmd.pus.s8\_fsfw\_action, 29
- tmtccmd.pus.s8\_fsfw\_action\_defs, 29
- tmtccmd.pus.tc.s200\_fsfw\_mode, 36
- tmtccmd.pus.tc.s20\_fsfw\_param, 33
- tmtccmd.pus.tc.s3\_fsfw\_hk, 25
- tmtccmd.pus.tc.s5\_event, 27
- tmtccmd.pus.tc.s8\_fsfw\_action, 29
- tmtccmd.pus.tm.s1\_verification, 25
- tmtccmd.pus.tm.s200\_fsfw\_mode, 36
- tmtccmd.pus.tm.s20\_fsfw\_param, 35
- tmtccmd.pus.tm.s2\_rawcmd, 25
- tmtccmd.pus.tm.s3\_fsfw\_hk, 25
- tmtccmd.pus.tm.s3\_hk\_base, 25
- tmtccmd.pus.tm.s5\_fsfw\_event, 28
- tmtccmd.pus.tm.s8\_fsfw\_action, 29
- tmtccmd.tmtc.ccsds\_seq\_sender, 50
- tmtccmd.tmtc.ccsds\_tm\_listener, 43
- tmtccmd.tmtc.common, 43
- tmtccmd.tmtc.handler, 46
- tmtccmd.tmtc.procedure, 49
- tmtccmd.tmtc.queue, 47
- tmtccmd.tmtc.tm\_base, 44
- tmtccmd.util, 68
- tmtccmd.util.conf\_util, 67
- tmtccmd.util.exit, 65
- tmtccmd.util.hammingcode, 65
- tmtccmd.util.json, 63
- tmtccmd.util.obj\_id, 64
- tmtccmd.version, 63



## A

- `acquire_timeout()` (in module `tmtccmd.util.conf_util`), 67
- `add()` (`tmtccmd.config.tmtc.OpCodeEntry` method), 57
- `add_apid_handler()` (`tmtccmd.tmtc.common.CcsdsTmHandler` method), 43
- `add_ccsds_tc()` (`tmtccmd.tmtc.queue.DefaultPusQueueHelper` method), 47
- `add_cfdp_args()` (`tmtccmd.config.args.PreArgsParsingWrapper` method), 54
- `add_cfdp_procedure_arguments()` (in module `tmtccmd.config.args`), 54
- `add_child()` (`tmtccmd.config.tmtc.CmdTreeNode` method), 56
- `add_def_proc_and_cfdp_as_subparsers()` (`tmtccmd.config.args.PreArgsParsingWrapper` method), 54
- `add_def_proc_args()` (`tmtccmd.config.args.PreArgsParsingWrapper` method), 54
- `add_default_com_if_arguments()` (in module `tmtccmd.config.args`), 54
- `add_default_tmtccmd_args()` (in module `tmtccmd.config.args`), 54
- `add_ethernet_arguments()` (in module `tmtccmd.config.args`), 55
- `add_generic_arguments()` (in module `tmtccmd.config.args`), 55
- `add_log_cmd()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `add_packet_delay()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `add_packet_delay_ms()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `add_pus_tc()` (`tmtccmd.tmtc.queue.DefaultPusQueueHelper` method), 47
- `add_raw_tc()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `add_service()` (`tmtccmd.config.tmtc.TmtcDefinitionWrapper` method), 57
- `add_tc()` (`tmtccmd.pus.VerificationWrapper` method), 24
- `add_tm()` (`tmtccmd.pus.VerificationWrapper` method), 24
- `add_tmtc_listener_arg()` (in module `tmtccmd.config.args`), 55
- `add_tmtc_mode_arguments()` (in module `tmtccmd.config.args`), 55
- `add_tree_commanding_arguments()` (in module `tmtccmd.config.args`), 55
- `add_wait()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `add_wait_ms()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `add_wait_seconds()` (`tmtccmd.tmtc.queue.QueueHelperBase` method), 48
- `AnsiColors` (class in `tmtccmd.util.conf_util`), 67
- `apid` (`tmtccmd.config.args.CommandingParams` attribute), 53
- `apid` (`tmtccmd.config.args.SetupParams` property), 54
- `apid` (`tmtccmd.tmtc.tm_base.PusTmBase` property), 44
- `apid` (`tmtccmd.tmtc.tm_base.PusTmInterface` property), 45
- `append_packet_info()` (`tmtccmd.tmtc.tm_base.PusTmInfoBase` method), 45
- `append_packet_info()` (`tmtccmd.tmtc.tm_base.PusTmInfoInterface` method), 45
- `append_telemetry_column_headers()` (`tmtccmd.pus.tm.s200_fsfw_mode.Service200FsfwTm` method), 37
- `append_telemetry_column_headers()` (`tmtccmd.tmtc.tm_base.PusTmInfoBase` method), 45
- `append_telemetry_column_headers()` (`tmtccmd.tmtc.tm_base.PusTmInfoInterface` method), 45

- method), 45
- append\_telemetry\_content() (tmtccmd.pus.tm.s200\_fsfw\_mode.Service200FsfwTm method), 37
- append\_telemetry\_content() (tmtccmd.tmtc.tm\_base.PusTmInfoBase method), 45
- append\_telemetry\_content() (tmtccmd.tmtc.tm\_base.PusTmInfoInterface method), 45
- AppParams (class in tmtccmd.config.args), 52
- args\_to\_all\_params\_for\_cfdp() (in module tmtccmd.config.args), 55
- args\_to\_all\_params\_tmtc() (in module tmtccmd.config.args), 55
- args\_to\_params\_generic() (in module tmtccmd.config.args), 55
- as\_hex\_string (tmtccmd.util.obj\_id.ComponentIdBase property), 64
- as\_u32() (tmtccmd.pus.s20\_fsfw\_param\_defs.ParameterId property), 32
- assign\_communication\_interface() (tmtccmd.config.hook.HookBase method), 51
- auto (tmtccmd.com.tcpip\_utils.TcpIpConfigIds attribute), 13
- ## B
- backend\_mode\_conversion() (in module tmtccmd.config), 51
- BackendParams (class in tmtccmd.config.args), 52
- BackendRequest (class in tmtccmd.core.base), 41
- base\_tm (tmtccmd.pus.tm.s20\_fsfw\_param.Service20ParamDumpWrapper property), 35
- baud\_rate (tmtccmd.com.serial\_base.SerialCfg attribute), 17
- bit\_extractor() (in module tmtccmd.fsfw), 68
- bit\_extractor() (tmtccmd.fsfw.tmtc\_printer.FsfwTmTcPrinter static method), 66
- BLANK (tmtccmd.config.tmtc.TreePart attribute), 58
- BLUE (tmtccmd.util.conf\_util.AnsiColors attribute), 67
- BOLD (tmtccmd.util.conf\_util.AnsiColors attribute), 67
- build\_from\_tc() (tmtccmd.pus.s11\_tc\_sched\_defs.TcSchedReqId class method), 30
- BUSY (tmtccmd.tmtc.ccsds\_seq\_sender.SenderMode attribute), 50
- bytes() (tmtccmd.com.qemu.DataFrame method), 20
- ## C
- call\_all\_definitions\_providers() (in module tmtccmd.config.tmtc), 58
- CALL\_NEXT (tmtccmd.core.base.BackendRequest attribute), 41
- CCSDS\_SPACE\_PACKETS (tmtccmd.tmtc.common.TmTypes attribute), 44
- CCSDS\_TC (tmtccmd.tmtc.queue.TcQueueEntryType attribute), 49
- ccsds\_version (tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm property), 35
- ccsds\_version (tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm property), 28
- CcsdsTmHandler (class in tmtccmd.tmtc.common), 43
- CcsdsTmListener (class in tmtccmd.tmtc.ccsds\_tm\_listener), 43
- CcsdsTmtcBackend (class in tmtccmd.core.ccsds\_backend), 40
- CFDP (tmtccmd.tmtc.procedure.TcProcedureType attribute), 49
- cfdp\_args\_to\_cfdp\_params() (in module tmtccmd.config.args), 55
- cfdp\_params() (tmtccmd.config.args.ProcedureParamsWrapper method), 54
- cfdp\_put\_req\_params\_to\_procedure() (in module tmtccmd.config), 51
- cfdp\_req\_to\_put\_req\_get\_req() (in module tmtccmd.config.cfdp), 58
- cfdp\_req\_to\_put\_req\_proxy\_put\_req() (in module tmtccmd.config.cfdp), 58
- cfdp\_req\_to\_put\_req\_regular() (in module tmtccmd.config.cfdp), 58
- cfdp\_request\_type (tmtccmd.tmtc.procedure.CfdpProcedure property), 49
- CfdpParams (class in tmtccmd.config.defs), 60
- CfdpProcedure (class in tmtccmd.tmtc.procedure), 49
- CfdpRequestBase (class in tmtccmd.cfdp.request), 38
- CfdpRequestWrapper (class in tmtccmd.cfdp.request), 38
- check\_args\_in\_dict() (in module tmtccmd.util.conf\_util), 67
- check\_json\_file() (in module tmtccmd.util.json), 63
- check\_port\_validity() (in module tmtccmd.com.ser\_utils), 18
- chunks() (tmtccmd.fsfw.tmtc\_printer.FsfwTmTcPrinter static method), 66
- clear\_layer\_is\_last\_child() (tmtccmd.config.tmtc.DepthInfo method), 57
- close() (tmtccmd.com.ComInterface method), 10
- close() (tmtccmd.com.dummy.DummyComIF method), 18
- close() (tmtccmd.com.qemu.QEMUComIF method), 20
- close() (tmtccmd.com.qemu.QmpConnection method), 21
- close() (tmtccmd.com.qemu.Usart method), 22

`close()` (*tmtccmd.com.serial\_cobs.SerialCobsComIF method*), 15  
`close()` (*tmtccmd.com.serial\_dle.SerialDleComIF method*), 16  
`close()` (*tmtccmd.com.tcp.TcpSpacepacketsClient method*), 11  
`close()` (*tmtccmd.com.udp.UdpClient method*), 13  
`close_com_if()` (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend method*), 40  
`close_port()` (*tmtccmd.com.serial\_base.SerialComBase method*), 17  
`closure_requested` (*tmtccmd.config.defs.CfdpParams attribute*), 60  
`cmd_path` (*tmtccmd.config.defs.TreeCommandingParams attribute*), 61  
`CmdTreeNode` (*class in tmtccmd.config.tmtc*), 56  
`COBS` (*tmtccmd.com.serial\_base.SerialCommunicationType attribute*), 17  
`CODE_OKAY` (*tmtccmd.util.hammingcode.HammingReturnCodes attribute*), 65  
`columns` (*tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId attribute*), 31  
`columns` (*tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter property*), 31  
`com_if` (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend property*), 40  
`COM_IF` (*tmtccmd.util.json.JsonKeyNames attribute*), 63  
`com_if_active()` (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend method*), 40  
`com_if_id` (*tmtccmd.com.serial\_base.SerialCfg attribute*), 17  
`com_if_id` (*tmtccmd.config.args.BackendParams attribute*), 52  
`com_if_id` (*tmtccmd.config.args.SetupParams property*), 54  
`com_if_id` (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend property*), 40  
`ComCfgBase` (*class in tmtccmd.config.com*), 59  
`ComInterface` (*class in tmtccmd.com*), 10  
`CommandingParams` (*class in tmtccmd.config.args*), 53  
`compl_style` (*tmtccmd.config.args.AppParams attribute*), 52  
`ComponentIdBase` (*class in tmtccmd.util.obj\_id*), 64  
`ComponentIdU16` (*class in tmtccmd.util.obj\_id*), 64  
`ComponentIdU32` (*class in tmtccmd.util.obj\_id*), 64  
`ComponentIdU8` (*class in tmtccmd.util.obj\_id*), 64  
`connection_lost()` (*tmtccmd.com.qemu.QmpProtocol method*), 22  
`connection_lost()` (*tmtccmd.com.qemu.UsartProtocol method*), 23  
`connection_made()` (*tmtccmd.com.qemu.QmpProtocol method*), 22  
`connection_made()` (*tmtccmd.com.qemu.UsartProtocol method*), 23  
`cont()` (*tmtccmd.com.qemu.QmpConnection method*), 22  
`contains_mode()` (*tmtccmd.pus.tm.s200\_fsfw\_mode.Service200FsfwReader method*), 37  
`contains_path()` (*tmtccmd.config.tmtc.CmdTreeNode method*), 56  
`contains_path_from_node_list()` (*tmtccmd.config.tmtc.CmdTreeNode method*), 56  
`CoreComInterfaces` (*class in tmtccmd.config.defs*), 60  
`CoreModeConverter` (*class in tmtccmd.config.defs*), 61  
`CoreModeList` (*class in tmtccmd.config.defs*), 61  
`CORNER` (*tmtccmd.config.tmtc.TreePart attribute*), 58  
`create_action_cmd()` (*in module tmtccmd.pus.tc.s8\_fsfw\_action*), 29  
`create_announce_mode_command()` (*in module tmtccmd.pus.tc.s200\_fsfw\_mode*), 36  
`create_announce_mode_recursive_command()` (*in module tmtccmd.pus.tc.s200\_fsfw\_mode*), 36  
`create_async()` (*tmtccmd.com.qemu.Usart static method*), 22  
`create_com_interface_cfg_default()` (*in module tmtccmd.config.com*), 59  
`create_com_interface_default()` (*in module tmtccmd.config.com*), 59  
`create_default_args_parser()` (*in module tmtccmd.config.args*), 55  
`create_default_parent_parser()` (*tmtccmd.config.args.PreArgsParsingWrapper method*), 54  
`create_default_parser()` (*tmtccmd.config.args.PreArgsParsingWrapper method*), 54  
`create_default_serial_interface()` (*in module tmtccmd.config.com*), 59  
`create_default_tcpip_interface()` (*in module tmtccmd.config.com*), 59  
`create_default_tmtc_backend()` (*in module tmtccmd*), 38  
`create_disable_event_reporting_command()` (*in module tmtccmd.pus.tc.s5\_event*), 27  
`create_disable_periodic_hk_command()` (*in module tmtccmd.pus.tc.s3\_fsfw\_hk*), 25  
`create_disable_periodic_hk_command_with_diag()` (*in module tmtccmd.pus.tc.s3\_fsfw\_hk*), 25  
`create_dump_param_cmd()` (*in module tmtccmd.pus.tc.s20\_fsfw\_param*), 33  
`create_enable_event_reporting_command()` (*in module tmtccmd.pus.tc.s5\_event*), 27  
`create_enable_periodic_hk_command()` (*in module tmtccmd.pus.tc.s3\_fsfw\_hk*), 25

[create\\_enable\\_periodic\\_hk\\_command\\_with\\_diag\(\)](#) *CUSTOM* (*tmtccmd.tmtc.queue.TcQueueEntryType* attribute), 49  
 (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_enable\\_periodic\\_hk\\_command\\_with\\_interval\(\)](#) *CustomFswPusService* (class in *tmtccmd.pus*), 24  
 (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_enable\\_periodic\\_hk\\_command\\_with\\_interval\\_with\\_diag\(\)](#) *CustomProcedureInfo* (class in *tmtccmd.pus.tc.procedure*), 49  
 (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_load\\_param\\_cmd\(\)](#) (in module *tmtccmd.pus.tc.s20\_fsw\_param*), 33  
[create\\_load\\_param\\_cmd\\_from\\_raw\(\)](#) (in module *tmtccmd.pus.tc.s20\_fsw\_param*), 33  
[create\\_matrix\\_double\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 32  
[create\\_matrix\\_float\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 32  
[create\\_mode\\_command\(\)](#) (in module *tmtccmd.pus.tc.s200\_fsw\_mode*), 36  
[create\\_modify\\_collection\\_interval\\_cmd\(\)](#) (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_modify\\_collection\\_interval\\_cmd\\_with\\_diag\(\)](#) (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_read\\_mode\\_command\(\)](#) (in module *tmtccmd.pus.tc.s200\_fsw\_mode*), 36  
[create\\_request\\_one\\_diag\\_command\(\)](#) (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_request\\_one\\_hk\\_command\(\)](#) (in module *tmtccmd.pus.tc.s3\_fsw\_hk*), 26  
[create\\_scalar\\_boolean\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 32  
[create\\_scalar\\_double\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 32  
[create\\_scalar\\_float\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 32  
[create\\_scalar\\_i16\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 32  
[create\\_scalar\\_i32\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[create\\_scalar\\_i8\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[create\\_scalar\\_u16\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[create\\_scalar\\_u32\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[create\\_scalar\\_u8\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[create\\_vector\\_double\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[create\\_vector\\_float\\_parameter\(\)](#) (in module *tmtccmd.pus.s20\_fsw\_param\_defs*), 33  
[cross\\_mark\\_unicode\(\)](#) (in module *tmtccmd.pus*), 24  
[current\\_procedure](#) (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend* property), 40  
*CUSTOM* (*tmtccmd.tmtc.procedure.TcProcedureType* attribute), 50

**D**  
[dash\\_unicode\(\)](#) (in module *tmtccmd.pus*), 24  
[data\\_available\(\)](#) (*tmtccmd.com.ComInterface* method), 10  
[data\\_available\(\)](#) (*tmtccmd.cmd.com.dummy.DummyComIF* method), 18  
[data\\_available\(\)](#) (*tmtccmd.com.qemu.QEMUComIF* method), 20  
[data\\_available\(\)](#) (*tmtccmd.cmd.com.serial\_cobs.SerialCobsComIF* method), 15  
[data\\_available\(\)](#) (*tmtccmd.cmd.com.serial\_dle.SerialDleComIF* method), 16  
[data\\_available\(\)](#) (*tmtccmd.cmd.com.tcp.TcpSpacepacketsClient* method), 11  
[data\\_available\(\)](#) (*tmtccmd.com.udp.UdpClient* method), 13  
[data\\_available\\_dle\(\)](#) (*tmtccmd.cmd.com.qemu.QEMUComIF* method), 21  
[data\\_available\\_fixed\\_frame\(\)](#) (*tmtccmd.cmd.com.qemu.QEMUComIF* method), 21  
[data\\_available\\_from\\_queue\(\)](#) (*tmtccmd.cmd.com.serial\_base.SerialComBase* static method), 17  
[data\\_received\(\)](#) (*tmtccmd.com.qemu.QmpProtocol* method), 22  
[data\\_received\(\)](#) (*tmtccmd.com.qemu.UsartProtocol* method), 23  
[DataFrame](#) (class in *tmtccmd.com.qemu*), 20  
[date\\_suffix\(\)](#) (in module *tmtccmd.logging.pus*), 63  
[default\\_json\\_path\(\)](#) (in module *tmtccmd.config.defs*), 61  
[default\\_operation\(\)](#) (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend* method), 40  
[default\\_serial\\_cfg\\_baud\\_and\\_port\\_setup\(\)](#) (in module *tmtccmd.config.com*), 59



default\_tcpip\_cfg\_setup() (in module *tmtccmd.config.com*), 60  
 default\_toml\_path() (in module *tmtccmd.config.defs*), 61  
 DefaultApidHandler (class in *tmtccmd.tmtc.common*), 44  
 DefaultPusQueueHelper (class in *tmtccmd.tmtc.queue*), 47  
 DEFINITIONS (*tmtccmd.pus.tm.s3\_hk\_base.HkContentType* attribute), 25  
 delay (*tmtccmd.config.args.CommandingParams* attribute), 53  
 DELAY\_CUSTOM (*tmtccmd.core.base.BackendRequest* attribute), 41  
 DELAY\_IDLE (*tmtccmd.core.base.BackendRequest* attribute), 41  
 DELAY\_LISTENER (*tmtccmd.core.base.BackendRequest* attribute), 41  
 DepthInfo (class in *tmtccmd.config.tmtc*), 57  
 deserialize\_scalar\_entry() (in module *tmtccmd.pus.s20\_fsfw\_param\_defs*), 33  
 dest\_file (*tmtccmd.config.defs.CfdpParams* attribute), 60  
 determine\_baud\_rate() (in module *tmtccmd.com.ser\_utils*), 18  
 determine\_cmd\_path() (in module *tmtccmd.config.args*), 55  
 determine\_com\_if() (in module *tmtccmd.com.utils*), 20  
 determine\_com\_port() (in module *tmtccmd.com.ser\_utils*), 18  
 determine\_recv\_buffer\_len() (in module *tmtccmd.com.tcpip\_utils*), 13  
 determine\_tcp\_send\_address() (in module *tmtccmd.com.tcpip\_utils*), 13  
 determine\_tcpip\_address() (in module *tmtccmd.com.tcpip\_utils*), 13  
 determine\_udp\_recv\_address() (in module *tmtccmd.com.tcpip\_utils*), 13  
 determine\_udp\_send\_address() (in module *tmtccmd.com.tcpip\_utils*), 13  
 disable\_periodic\_hk\_command() (in module *tmtccmd.pus.tc.s3\_fsfw\_hk*), 26  
 DisplayMode (class in *tmtccmd.fsfw.tmtc\_printer*), 66  
 DLE\_ENCODING (*tmtccmd.com.serial\_base.SerialCommunicationType* attribute), 17  
 dle\_max\_frame (*tmtccmd.com.serial\_dle.DleCfg* attribute), 16  
 dle\_queue\_len (*tmtccmd.com.serial\_dle.DleCfg* attribute), 16  
 DleCfg (class in *tmtccmd.com.serial\_dle*), 16  
 dlog() (*tmtccmd.pus.VerificationWrapper* method), 24  
 domain\_id (*tmtccmd.pus.s20\_fsfw\_param\_defs.ParameterId* attribute), 32  
 DONE (*tmtccmd.tmtc.ccsds\_seq\_sender.SenderMode* attribute), 50  
 DUMMY (*tmtccmd.config.defs.CoreComInterfaces* attribute), 60  
 DummyComIF (class in *tmtccmd.com.dummy*), 18  
 DummyHandler (class in *tmtccmd.com.dummy*), 19  
 E  
 EDGE (*tmtccmd.config.tmtc.TreePart* attribute), 58  
 empty() (*tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter* class method), 31  
 empty() (*tmtccmd.pus.s20\_fsfw\_param\_defs.ParameterId* class method), 32  
 empty() (*tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm* class method), 35  
 empty() (*tmtccmd.pus.tm.s5\_fsfw\_event.EventDefinition* class method), 28  
 empty() (*tmtccmd.tmtc.procedure.TreeCommandingProcedure* class method), 50  
 empty() (*tmtccmd.tmtc.queue.QueueHelperBase* method), 48  
 empty() (*tmtccmd.tmtc.queue.QueueWrapper* class method), 48  
 enable\_periodic\_hk\_command() (in module *tmtccmd.pus.tc.s3\_fsfw\_hk*), 26  
 enable\_periodic\_hk\_command\_with\_interval() (in module *tmtccmd.pus.tc.s3\_fsfw\_hk*), 26  
 encode\_cr (*tmtccmd.com.serial\_dle.DleCfg* attribute), 16  
 entry\_type (*tmtccmd.tmtc.queue.QueueEntryHelper* property), 48  
 ERROR\_ECC (*tmtccmd.util.hammingcode.HammingReturnCodes* attribute), 65  
 ERROR\_MULTI\_BIT (*tmtccmd.util.hammingcode.HammingReturnCodes* attribute), 65  
 ERROR\_SINGLE\_BIT (*tmtccmd.util.hammingcode.HammingReturnCodes* attribute), 65  
 EthAddr (class in *tmtccmd.com.tcpip\_utils*), 12  
 event\_definition (*tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm* property), 28  
 event\_id (*tmtccmd.pus.tm.s5\_fsfw\_event.EventDefinition* attribute), 28  
 EventDefinition (class in *tmtccmd.pus.tm.s5\_fsfw\_event*), 28  
 EventInfo (class in *tmtccmd.pus.s5\_fsfw\_event\_defs*), 28  
 EventSeverity (class in *tmtccmd.pus.s5\_satrs\_event\_defs*), 27  
 EventU32 (class in *tmtccmd.pus.s5\_satrs\_event\_defs*), 27  
 exit\_gracefully() (*tmtccmd.util.exit.GracefulKiller* method), 65

extract\_subnode() (tmtc-  
     cmd.config.tmtc.CmdTreeNode method), 56  
 extract\_subnode\_by\_node\_list() (tmtc-  
     cmd.config.tmtc.CmdTreeNode method), 56  
**F**  
 feed\_cb() (tmtccmd.tmtc.handler.TcHandlerBase  
     method), 46  
 FeedWrapper (class in tmtccmd.tmtc.handler), 46  
 file\_location (tmtc-  
     cmd.pus.s5\_fsw\_event\_defs.EventInfo at-  
     tribute), 28  
 find\_com\_port\_from\_hint() (in module tmtc-  
     cmd.com.ser\_utils), 18  
 flush() (tmtccmd.com.qemu.Usart method), 22  
 from\_bytes() (tmtccmd.pus.tm.s5\_fsw\_event.EventDefinition  
     class method), 28  
 from\_bytes\_typed() (tmtc-  
     cmd.util.obj\_id.ComponentIdU16 class  
     method), 64  
 from\_bytes\_typed() (tmtc-  
     cmd.util.obj\_id.ComponentIdU32 class  
     method), 64  
 from\_bytes\_typed() (tmtc-  
     cmd.util.obj\_id.ComponentIdU8 class method),  
     64  
 from\_millis() (tmtccmd.tmtc.queue.PacketDelayEntry  
     class method), 47  
 from\_millis() (tmtccmd.tmtc.queue.WaitEntry class  
     method), 49  
 FROM\_TIMETAG (tmtccmd.pus.s11\_tc\_sched\_defs.TypeOfTimeWindow  
     attribute), 30  
 FROM\_TIMETAG\_TO\_TIMETAG (tmtc-  
     cmd.pus.s11\_tc\_sched\_defs.TypeOfTimeWindow  
     attribute), 30  
 from\_tm() (tmtccmd.pus.tm.s20\_fsw\_param.Service20FswTm  
     class method), 35  
 from\_tm() (tmtccmd.pus.tm.s5\_fsw\_event.Service5Tm  
     class method), 29  
 from\_tuple() (tmtccmd.com.tcpip\_utils.EthAddr class  
     method), 12  
 FrontendBase (class in tmtccmd.core.base), 42  
 fsfw\_param\_id (tmtc-  
     cmd.pus.s20\_fsw\_param\_defs.Parameter  
     attribute), 31  
 FswParamId (class in tmtc-  
     cmd.pus.s20\_fsw\_param\_defs), 31  
 FswTmTcPrinter (class in tmtccmd.fsw.tmtc\_printer),  
     66  
**G**  
 gen\_console\_char\_from\_status() (in module tmtc-  
     cmd.pus), 25  
 gen\_file\_char\_from\_status() (in module tmtc-  
     cmd.pus), 25  
 generate\_one\_diag\_command() (in module tmtc-  
     cmd.pus.tc.s3\_fsw\_hk), 26  
 generate\_one\_hk\_command() (in module tmtc-  
     cmd.pus.tc.s3\_fsw\_hk), 26  
 generate\_reply\_package() (tmtc-  
     cmd.com.dummy.DummyHandler method),  
     19  
 generic\_cfdp\_params\_to\_put\_request() (in mod-  
     ule tmtccmd.config.cfdp), 58  
 generic\_hk\_tm\_print() (tmtc-  
     cmd.fsw.tmtc\_printer.FswTmTcPrinter  
     method), 66  
 generic\_short\_string() (tmtc-  
     cmd.fsw.tmtc\_printer.FswTmTcPrinter static  
     method), 66  
 GenericApidHandlerBase (class in tmtc-  
     cmd.tmtc.common), 44  
 get\_base\_component\_id\_mapping() (in module tmtc-  
     cmd.config.objects), 61  
 get\_cmd\_history() (tmtccmd.config.hook.HookBase  
     method), 52  
 get\_com\_if\_dict() (tmtccmd.config.hook.HookBase  
     method), 52  
 get\_command\_definitions() (tmtc-  
     cmd.config.hook.HookBase method), 52  
 get\_communication\_interface() (tmtc-  
     cmd.config.hook.HookBase method), 52  
 get\_core\_object\_ids() (in module tmtc-  
     cmd.config.objects), 61  
 get\_current\_tmtc\_file\_name() (tmtc-  
     cmd.logging.pus.RegularTmtcLogWrapper  
     class method), 62  
 get\_custom\_printout() (tmtc-  
     cmd.tmtc.tm\_base.PusTmInfoBase method),  
     45  
 get\_custom\_printout() (tmtc-  
     cmd.tmtc.tm\_base.PusTmInfoInterface  
     method), 45  
 get\_data\_in\_waiting() (tmtccmd.com.qemu.Usart  
     method), 22  
 get\_default\_descript\_txt() (in module tmtc-  
     cmd.config.args), 55  
 get\_global() (in module tmtc-  
     cmd.core.globals\_manager), 42  
 get\_lib\_logger() (in module tmtccmd), 38  
 get\_object\_ids() (tmtccmd.config.hook.HookBase  
     method), 52  
 get\_param() (tmtccmd.pus.tm.s20\_fsw\_param.Service20ParamDumpWr  
     method), 35  
 get\_print\_info() (tmtc-  
     cmd.tmtc.tm\_base.PusTmInfoBase method),

45  
get\_print\_info() (tmtccmd.tmtc.tm\_base.PusTmInfoInterface method), 45  
get\_source\_data\_string() (tmtccmd.tmtc.tm\_base.PusTmInfoBase method), 45  
get\_source\_data\_string() (tmtccmd.tmtc.tm\_base.PusTmInfoInterface method), 45  
get\_str() (tmtccmd.config.defs.CoreModeConverter static method), 61  
get\_tmtc\_definitions() (tmtccmd.config.hook.HookBase method), 52  
get\_type() (tmtccmd.tmtc.common.TmHandlerBase method), 44  
get\_validity\_buffer\_str() (in module tmtccmd.fsfw.tmtc\_printer), 67  
get\_validity\_buffer\_str() (tmtccmd.fsfw.tmtc\_printer.FsfwTmTcPrinter static method), 67  
get\_version() (in module tmtccmd.version), 63  
GracefulKiller (class in tmtccmd.util.exit), 65  
GREEN (tmtccmd.util.conf\_util.AnsiColors attribute), 67  
group\_id (tmtccmd.pus.s5\_satrs\_event\_defs.EventU32 attribute), 27

## H

hamming\_compute\_256() (in module tmtccmd.util.hammingcode), 65  
hamming\_compute\_256x() (in module tmtccmd.util.hammingcode), 65  
hamming\_test() (in module tmtccmd.util.hammingcode), 65  
hamming\_verify\_256() (in module tmtccmd.util.hammingcode), 66  
hamming\_verify\_256x() (in module tmtccmd.util.hammingcode), 66  
HammingReturnCodes (class in tmtccmd.util.hammingcode), 65  
handle\_long\_tm\_print() (tmtccmd.fsfw.tmtc\_printer.FsfwTmTcPrinter method), 67  
handle\_new\_queue\_forced() (tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender method), 50  
handle\_non\_tc\_entry() (tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender method), 50  
handle\_packet() (tmtccmd.tmtc.common.CcsdsTmHandler method), 43  
handle\_tm() (tmtccmd.tmtc.common.DefaultApidHandler method), 44  
handle\_tm() (tmtccmd.tmtc.common.GenericApidHandlerBase method), 44  
handle\_tm() (tmtccmd.tmtc.common.SpecificApidHandlerBase method), 44  
has\_apid() (tmtccmd.tmtc.common.CcsdsTmHandler method), 44  
has\_custom\_hk\_handling (tmtccmd.pus.tm.s3\_hk\_base.Service3Base property), 25  
HIGH (tmtccmd.pus.s5\_fsfw\_event\_defs.Severity attribute), 28  
HIGH (tmtccmd.pus.s5\_satrs\_event\_defs.EventSeverity attribute), 27  
HK (tmtccmd.pus.tm.s3\_hk\_base.HkContentType attribute), 25  
hk\_definitions\_list (tmtccmd.pus.tm.s3\_hk\_base.Service3Base property), 25  
HkContentType (class in tmtccmd.pus.tm.s3\_hk\_base), 25  
HookBase (class in tmtccmd.config.hook), 51

## I

id (tmtccmd.com.ComInterface property), 10  
id (tmtccmd.com.dummy.DummyComIF property), 19  
id (tmtccmd.com.qemu.QEMUComIF property), 21  
id (tmtccmd.com.serial\_cobs.SerialCobsComIF property), 15  
id (tmtccmd.com.serial\_dle.SerialDleComIF property), 16  
id (tmtccmd.com.tcp.TcpSpacepacketsClient property), 12  
id (tmtccmd.com.udp.UdpClient property), 14  
id (tmtccmd.pus.s5\_fsfw\_event\_defs.EventInfo attribute), 28  
id\_u64 (tmtccmd.pus.s11\_tc\_sched\_defs.TcSchedReqId property), 30  
IDLE (tmtccmd.config.defs.CoreModeList attribute), 61  
IDLE (tmtccmd.core.base.TcMode attribute), 42  
IDLE (tmtccmd.core.base.TmMode attribute), 42  
info (tmtccmd.pus.s5\_fsfw\_event\_defs.EventInfo attribute), 28  
INFO (tmtccmd.pus.s5\_fsfw\_event\_defs.Severity attribute), 28  
INFO (tmtccmd.pus.s5\_satrs\_event\_defs.EventSeverity attribute), 27  
info() (tmtccmd.config.tmtc.OpCodeEntry method), 57  
init\_logger() (in module tmtccmd), 38  
init\_printout() (in module tmtccmd), 39  
initialize() (tmtccmd.com.ComInterface method), 10  
initialize() (tmtccmd.com.dummy.DummyComIF method), 19  
initialize() (tmtccmd.com.qemu.QEMUComIF method), 21

initialize() (*tmtccmd.com.serial\_cobs.SerialCobsComIF* method), 15  
 initialize() (*tmtccmd.com.serial\_dle.SerialDleComIF* method), 16  
 initialize() (*tmtccmd.com.tcp.TcpSpacepacketsClient* method), 12  
 initialize() (*tmtccmd.com.udp.UdpClient* method), 14  
 inject\_frame\_error() (*tmtccmd.com.qemu.Usart* method), 22  
 inject\_overrun\_error() (*tmtccmd.com.qemu.Usart* method), 22  
 inject\_parity\_error() (*tmtccmd.com.qemu.Usart* method), 22  
 inject\_timeout\_error() (*tmtccmd.com.qemu.Usart* method), 23  
 insert\_telecommand() (*tmtccmd.com.dummy.DummyHandler* method), 19  
 inter\_cmd\_delay (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend* property), 40  
 interactive (*tmtccmd.config.args.BackendParams* attribute), 52  
 ip\_addr (*tmtccmd.com.tcpip\_utils.EthAddr* attribute), 12  
 is\_cant\_reach\_mode\_reply() (*tmtccmd.pus.tm.s200\_fsfw\_mode.Service200FsfwReader* method), 37  
 is\_layer\_for\_last\_child() (*tmtccmd.config.tmtc.DepthInfo* method), 57  
 is\_leaf() (*tmtccmd.config.tmtc.CmdTreeNode* method), 56  
 is\_open() (*tmtccmd.com.ComInterface* method), 10  
 is\_open() (*tmtccmd.com.dummy.DummyComIF* method), 19  
 is\_open() (*tmtccmd.com.qemu.QEMUComIF* method), 21  
 is\_open() (*tmtccmd.com.serial\_cobs.SerialCobsComIF* method), 15  
 is\_open() (*tmtccmd.com.serial\_dle.SerialDleComIF* method), 16  
 is\_open() (*tmtccmd.com.tcp.TcpSpacepacketsClient* method), 12  
 is\_open() (*tmtccmd.com.udp.UdpClient* method), 14  
 is\_port\_open() (*tmtccmd.com.serial\_base.SerialComBase* method), 17  
 is\_tc (*tmtccmd.tmtc.queue.QueueEntryHelper* property), 48  
 is\_tc() (*tmtccmd.tmtc.queue.TcQueueEntryBase* method), 48

## J

JsonKeyNames (class in *tmtccmd.util.json*), 63

## K

keyboard\_interrupt\_handler() (in module *tmtccmd.util.exit*), 65  
 kill\_now (*tmtccmd.util.exit.GracefulKiller* attribute), 65

## L

LINE (*tmtccmd.config.tmtc.TreePart* attribute), 58  
 linear\_index (*tmtccmd.pus.s20\_fsfw\_param\_defs.ParameterId* attribute), 32  
 listener (*tmtccmd.config.args.BackendParams* attribute), 52  
 LISTENER (*tmtccmd.core.base.TmMode* attribute), 42  
 LISTENER\_MODE (*tmtccmd.config.defs.CoreModeList* attribute), 61  
 lock\_global\_pool() (in module *tmtccmd.core.globals\_manager*), 42  
 LOG (*tmtccmd.tmtc.queue.TcQueueEntryType* attribute), 49  
 log\_bytes\_readable() (*tmtccmd.logging.pus.RawTmtcLogBase* method), 62  
 log\_bytes\_repr() (*tmtccmd.logging.pus.RawTmtcLogBase* method), 62  
 log\_progress\_to\_console\_from\_status() (*tmtccmd.pus.VerificationWrapper* method), 24  
 log\_repr() (*tmtccmd.logging.pus.RawTmtcLogBase* method), 62  
 log\_tc() (*tmtccmd.logging.pus.RawTmtcLogBase* method), 62  
 log\_tm() (*tmtccmd.logging.pus.RawTmtcLogBase* method), 62  
 log\_to\_console() (*tmtccmd.pus.VerificationWrapper* method), 24  
 log\_to\_console\_from\_req\_id() (*tmtccmd.pus.VerificationWrapper* method), 24  
 log\_to\_file() (*tmtccmd.pus.VerificationWrapper* method), 24  
 log\_to\_file\_from\_req\_id() (*tmtccmd.pus.VerificationWrapper* method), 24  
 log\_to\_file\_from\_status() (*tmtccmd.pus.VerificationWrapper* method), 24  
 LogQueueEntry (class in *tmtccmd.tmtc.queue*), 47  
 LONG (*tmtccmd.fsfw.tmtc\_printer.DisplayMode* attribute), 66  
 LOW (*tmtccmd.pus.s5\_fsfw\_event\_defs.Severity* attribute), 28  
 LOW (*tmtccmd.pus.s5\_satrs\_event\_defs.EventSeverity* attribute), 27

## M

MAGNETA (*tmtccmd.util.conf\_util.AnsiColors* attribute), 67



make\_action\_id() (in module *tmtccmd.pus.tc.s8\_fsfw\_action*), 29  
 make\_fsfw\_action\_cmd() (in module *tmtccmd.pus.tc.s8\_fsfw\_action*), 29  
 make\_interval() (in module *tmtccmd.pus.tc.s3\_fsfw\_hk*), 26  
 make\_sid() (in module *tmtccmd.pus.tc.s3\_fsfw\_hk*), 26  
 MEDIUM (*tmtccmd.pus.s5\_fsfw\_event\_defs.Severity* attribute), 28  
 MEDIUM (*tmtccmd.pus.s5\_satrs\_event\_defs.EventSeverity* attribute), 27  
 Mode (class in *tmtccmd.pus.tc.s200\_fsfw\_mode*), 36  
 mode (*tmtccmd.config.args.BackendParams* attribute), 53  
 mode (*tmtccmd.config.args.SetupParams* property), 54  
 mode (*tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender* property), 50  
 mode\_to\_req() (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend* method), 40  
 ModeWrapper (class in *tmtccmd.core.base*), 42  
 modify\_collection\_interval() (in module *tmtccmd.pus.tc.s3\_fsfw\_hk*), 27  
 module  
     *tmtccmd*, 38  
     *tmtccmd.cfdp.request*, 38  
     *tmtccmd.com*, 10  
     *tmtccmd.com.dummy*, 18  
     *tmtccmd.com.qemu*, 20  
     *tmtccmd.com.ser\_utils*, 18  
     *tmtccmd.com.serial\_base*, 17  
     *tmtccmd.com.serial\_cobs*, 15  
     *tmtccmd.com.serial\_dle*, 16  
     *tmtccmd.com.tcp*, 11  
     *tmtccmd.com.tcpip\_utils*, 12  
     *tmtccmd.com.udp*, 13  
     *tmtccmd.com.utils*, 20  
     *tmtccmd.config*, 51  
     *tmtccmd.config.args*, 52  
     *tmtccmd.config.cfdp*, 58  
     *tmtccmd.config.com*, 59  
     *tmtccmd.config.defs*, 60  
     *tmtccmd.config.hook*, 51  
     *tmtccmd.config.objects*, 61  
     *tmtccmd.config.tmtc*, 56  
     *tmtccmd.core*, 40  
     *tmtccmd.core.base*, 41  
     *tmtccmd.core.ccsds\_backend*, 40  
     *tmtccmd.core.globals\_manager*, 42  
     *tmtccmd.fsfw*, 68  
     *tmtccmd.fsfw.tmtc\_printer*, 66  
     *tmtccmd.logging.pus*, 62  
     *tmtccmd.pus*, 24  
     *tmtccmd.pus.s11\_tc\_sched*, 29  
     *tmtccmd.pus.s11\_tc\_sched\_defs*, 29  
     *tmtccmd.pus.s17\_test*, 30  
     *tmtccmd.pus.s17\_test\_defs*, 30  
     *tmtccmd.pus.s1\_verification*, 25  
     *tmtccmd.pus.s200\_fsfw\_mode*, 36  
     *tmtccmd.pus.s200\_fsfw\_mode\_defs*, 36  
     *tmtccmd.pus.s201\_fsfw\_health*, 37  
     *tmtccmd.pus.s201\_fsfw\_health\_defs*, 37  
     *tmtccmd.pus.s20\_fsfw\_param*, 31  
     *tmtccmd.pus.s20\_fsfw\_param\_defs*, 31  
     *tmtccmd.pus.s5\_fsfw\_event*, 28  
     *tmtccmd.pus.s5\_fsfw\_event\_defs*, 28  
     *tmtccmd.pus.s5\_satrs\_event*, 27  
     *tmtccmd.pus.s5\_satrs\_event\_defs*, 27  
     *tmtccmd.pus.s8\_fsfw\_action*, 29  
     *tmtccmd.pus.s8\_fsfw\_action\_defs*, 29  
     *tmtccmd.pus.tc.s200\_fsfw\_mode*, 36  
     *tmtccmd.pus.tc.s20\_fsfw\_param*, 33  
     *tmtccmd.pus.tc.s3\_fsfw\_hk*, 25  
     *tmtccmd.pus.tc.s5\_event*, 27  
     *tmtccmd.pus.tc.s8\_fsfw\_action*, 29  
     *tmtccmd.pus.tm.s1\_verification*, 25  
     *tmtccmd.pus.tm.s200\_fsfw\_mode*, 36  
     *tmtccmd.pus.tm.s20\_fsfw\_param*, 35  
     *tmtccmd.pus.tm.s2\_rawcmd*, 25  
     *tmtccmd.pus.tm.s3\_fsfw\_hk*, 25  
     *tmtccmd.pus.tm.s3\_hk\_base*, 25  
     *tmtccmd.pus.tm.s5\_fsfw\_event*, 28  
     *tmtccmd.pus.tm.s8\_fsfw\_action*, 29  
     *tmtccmd.tmtc.ccsds\_seq\_sender*, 50  
     *tmtccmd.tmtc.ccsds\_tm\_listener*, 43  
     *tmtccmd.tmtc.common*, 43  
     *tmtccmd.tmtc.handler*, 46  
     *tmtccmd.tmtc.procedure*, 49  
     *tmtccmd.tmtc.queue*, 47  
     *tmtccmd.tmtc.tm\_base*, 44  
     *tmtccmd.util*, 68  
     *tmtccmd.util.conf\_util*, 67  
     *tmtccmd.util.exit*, 65  
     *tmtccmd.util.hammingcode*, 65  
     *tmtccmd.util.json*, 63  
     *tmtccmd.util.obj\_id*, 64  
     *tmtccmd.version*, 63  
 MULTI\_INTERACTIVE\_QUEUE\_MODE (*tmtccmd.config.defs.CoreModeList* attribute), 61  
 MULTI\_QUEUE (*tmtccmd.core.base.TcMode* attribute), 42  
 N  
 name (*tmtccmd.pus.s5\_fsfw\_event\_defs.EventInfo* attribute), 28  
 name\_dict (*tmtccmd.config.tmtc.CmdTreeNode* property), 57  
 new\_data\_available() (*tmtccmd.com.qemu.Usart* method), 23

no\_delay\_remaining() (tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender method), 50

NONE (tmtccmd.core.base.BackendRequest attribute), 41

NONE (tmtccmd.tmtc.common.TmTypes attribute), 44

NORMAL (tmtccmd.pus.tc.s200\_fsfw\_mode.Mode attribute), 36

NoValidProcedureSet, 41

## O

obj\_id (tmtccmd.util.obj\_id.ComponentIdBase property), 64

object\_id (tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId attribute), 31

object\_id (tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter property), 31

object\_id (tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwParam property), 35

object\_id (tmtccmd.pus.tm.s3\_hk\_base.Service3Base property), 25

ObjectIdBase (in module tmtccmd.util.obj\_id), 64

ObjectIdDictT (in module tmtccmd.util.obj\_id), 64

ObjectIdMapping (in module tmtccmd.util.obj\_id), 64

ObjectIdU16 (in module tmtccmd.util.obj\_id), 64

ObjectIdU32 (in module tmtccmd.util.obj\_id), 64

ObjectIdU8 (in module tmtccmd.util.obj\_id), 65

OFF (tmtccmd.pus.tc.s200\_fsfw\_mode.Mode attribute), 36

ON (tmtccmd.pus.tc.s200\_fsfw\_mode.Mode attribute), 36

ONE\_QUEUE (tmtccmd.core.base.TcMode attribute), 42

ONE\_QUEUE\_MODE (tmtccmd.config.defs.CoreModeList attribute), 61

op\_code\_dict\_num\_keys (tmtccmd.config.tmtc.OpCodeEntry property), 57

op\_code\_dict\_str\_keys (tmtccmd.config.tmtc.OpCodeEntry property), 57

op\_code\_entry() (tmtccmd.config.tmtc.TmtcDefinitionWrapper method), 57

OpCodeEntry (class in tmtccmd.config.tmtc), 57

OpCodeOptionBase (class in tmtccmd.config.tmtc), 57

open() (tmtccmd.com.ComInterface method), 10

open() (tmtccmd.com.dummy.DummyComIF method), 19

open() (tmtccmd.com.qemu.QEMUComIF method), 21

open() (tmtccmd.com.qemu.QmpConnection method), 22

open() (tmtccmd.com.qemu.Usart method), 23

open() (tmtccmd.com.serial\_cobs.SerialCobsComIF method), 15

open() (tmtccmd.com.serial\_dle.SerialDleComIF method), 16

open() (tmtccmd.com.tcp.TcpSpacepacketsClient method), 12

open() (tmtccmd.com.udp.UdpClient method), 14

open\_com\_if() (tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend method), 40

open\_port() (tmtccmd.com.serial\_base.SerialComBase method), 17

operation() (tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender method), 50

operation() (tmtccmd.tmtc.ccsds\_tm\_listener.CcsdsTmListener method), 43

OTHER\_ERROR (tmtccmd.util.hammingcode.HammingReturnCodes attribute), 65

## P

pack() (tmtccmd.pus.s11\_tc\_sched\_defs.TcSchedReqId method), 30

pack() (tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId method), 31

pack() (tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter method), 32

pack() (tmtccmd.pus.s20\_fsfw\_param\_defs.ParameterId method), 32

pack() (tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm method), 35

pack() (tmtccmd.pus.tm.s5\_fsfw\_event.EventDefinition method), 28

pack() (tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm method), 29

pack() (tmtccmd.tmtc.tm\_base.PusTmBase method), 44

pack() (tmtccmd.tmtc.tm\_base.PusTmInterface method), 45

pack\_boolean\_parameter\_app\_data() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 33

pack\_disable\_event\_reporting\_command() (in module tmtccmd.pus.tc.s5\_event), 27

pack\_enable\_event\_reporting\_command() (in module tmtccmd.pus.tc.s5\_event), 27

pack\_mode\_command() (in module tmtccmd.pus.tc.s200\_fsfw\_mode), 36

pack\_mode\_data() (in module tmtccmd.pus.tc.s200\_fsfw\_mode), 36

pack\_parameter\_id() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 34

pack\_scalar\_boolean\_parameter\_app\_data() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 34

pack\_scalar\_double\_param\_app\_data() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 34

pack\_scalar\_float\_param\_app\_data() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 34

pack\_scalar\_u8\_parameter\_app\_data() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 34

pack\_type\_and\_matrix\_data() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 34  
 PACKET\_DELAY (tmtccmd.tmtc.queue.TcQueueEntryType attribute), 49  
 packet\_id (tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm property), 35  
 packet\_id (tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm property), 29  
 packet\_seq\_control (tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm property), 35  
 packet\_seq\_control (tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm property), 29  
 PacketDelayEntry (class in tmtccmd.tmtc.queue), 47  
 PacketsTooSmallForCclds, 43  
 param1 (tmtccmd.pus.tm.s5\_fsfw\_event.EventDefinition attribute), 28  
 param2 (tmtccmd.pus.tm.s5\_fsfw\_event.EventDefinition attribute), 28  
 param\_id (tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId attribute), 31  
 param\_id (tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter property), 32  
 param\_raw (tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter attribute), 32  
 Parameter (class in tmtccmd.pus.s20\_fsfw\_param\_defs), 31  
 ParameterId (class in tmtccmd.pus.s20\_fsfw\_param\_defs), 32  
 params (tmtccmd.config.SetupWrapper property), 51  
 params\_to\_procedure\_conversion() (in module tmtccmd.config), 51  
 parse() (tmtccmd.config.args.PreArgsParsingWrapper method), 54  
 parse\_dataframes() (in module tmtccmd.com.qemu), 23  
 parse\_default\_tmtccmd\_input\_arguments() (in module tmtccmd.config.args), 55  
 parse\_fsfw\_events\_csv() (in module tmtccmd.fsfw), 68  
 parse\_fsfw\_objects\_csv() (in module tmtccmd.fsfw), 68  
 parse\_fsfw\_returnvalues\_csv() (in module tmtccmd.fsfw), 68  
 parse\_scalar\_param() (in module tmtccmd.pus.s20\_fsfw\_param\_defs), 33  
 parse\_scalar\_param() (tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter method), 32  
 pass\_telecommand() (tmtccmd.com.dummy.DummyHandler method), 19  
 PER\_DAY (tmtccmd.logging.pus.TimedLogWhen attribute), 62  
 PER\_HOUR (tmtccmd.logging.pus.TimedLogWhen attribute), 62  
 PER\_MINUTE (tmtccmd.logging.pus.TimedLogWhen attribute), 62  
 PER\_SECOND (tmtccmd.logging.pus.TimedLogWhen attribute), 63  
 perform\_mode\_operation() (tmtccmd.config.hook.HookBase method), 52  
 perform\_tree\_printout() (in module tmtccmd.config.args), 56  
 periodic\_op() (tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend method), 40  
 pfc (tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId attribute), 31  
 pfc (tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter property), 32  
 poll\_dle\_packets() (tmtccmd.com.qemu.QEMUComIF method), 21  
 poll\_tm() (tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend method), 40  
 port (tmtccmd.com.tcpip\_utils.EthAddr attribute), 12  
 PostArgsParsingWrapper (class in tmtccmd.config.args), 53  
 pre\_add\_cb() (tmtccmd.tmtc.queue.DefaultPusQueueHelper method), 47  
 pre\_add\_cb() (tmtccmd.tmtc.queue.QueueHelperBase method), 48  
 PreArgsParsingWrapper (class in tmtccmd.config.args), 53  
 prepare\_param\_packet\_header() (in module tmtccmd.pus.tc.s20\_fsfw\_param), 35  
 print\_data() (tmtccmd.fsfw.tmtc\_printer.FsfwTmTcPrinter static method), 67  
 print\_tree (tmtccmd.config.args.CommandingParams attribute), 53  
 print\_validity\_buffer() (tmtccmd.fsfw.tmtc\_printer.FsfwTmTcPrinter method), 67  
 proc\_type (tmtccmd.tmtc.procedure.ProcedureWrapper property), 49  
 ProcedureParamsWrapper (class in tmtccmd.config.args), 54  
 ProcedureWrapper (class in tmtccmd.tmtc.procedure), 49  
 prompt\_com\_if() (in module tmtccmd.com.utils), 20  
 prompt\_com\_port() (in module tmtccmd.com.ser\_utils), 18  
 prompt\_ip\_address() (in module tmtccmd.com.tcpip\_utils), 13  
 prompt\_recv\_buffer\_len() (in module tmtccmd.com.tcpip\_utils), 13  
 proxy\_op (tmtccmd.config.defs.CfdpParams attribute),

- 60
- ptc (*tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId* attribute), 31
- ptc (*tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter* property), 32
- ptype (*tmtccmd.config.args.ProcedureParamsWrapper* property), 54
- PUS\_TC (*tmtccmd.tmtc.queue.TcQueueEntryType* attribute), 49
- PusTcEntry (class in *tmtccmd.tmtc.queue*), 48
- PusTmBase (class in *tmtccmd.tmtc.tm\_base*), 44
- PusTmInfoBase (class in *tmtccmd.tmtc.tm\_base*), 45
- PusTmInfoInterface (class in *tmtccmd.tmtc.tm\_base*), 45
- PusTmInterface (class in *tmtccmd.tmtc.tm\_base*), 45
- PutRequestCfgWrapper (class in *tmtccmd.cfdp.request*), 38
- ## Q
- QEMUComIF (class in *tmtccmd.com.qemu*), 20
- QmpConnection (class in *tmtccmd.com.qemu*), 21
- QmpException, 22
- QmpProtocol (class in *tmtccmd.com.qemu*), 22
- queue\_finished\_cb() (*tmtccmd.tmtc.handler.TcHandlerBase* method), 47
- queue\_wrapper (*tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender* property), 50
- QueueEntryHelper (class in *tmtccmd.tmtc.queue*), 48
- QueueHelperBase (class in *tmtccmd.tmtc.queue*), 48
- QueueWrapper (class in *tmtccmd.tmtc.queue*), 48
- quit() (*tmtccmd.com.qemu.QmpConnection* method), 22
- ## R
- RAW (*tmtccmd.pus.tc.s200\_fsfw\_mode.Mode* attribute), 36
- RAW\_TC (*tmtccmd.tmtc.queue.TcQueueEntryType* attribute), 49
- RawTcEntry (class in *tmtccmd.tmtc.queue*), 48
- RawTmtcLogBase (class in *tmtccmd.logging.pus*), 62
- RawTmtcRotatingLogWrapper (class in *tmtccmd.logging.pus*), 62
- RawTmtcTimedLogWrapper (class in *tmtccmd.logging.pus*), 62
- read() (*tmtccmd.com.qemu.Usart* method), 23
- read\_async() (*tmtccmd.com.qemu.Usart* method), 23
- read\_until\_async() (*tmtccmd.com.qemu.Usart* method), 23
- receive() (*tmtccmd.com.ComInterface* method), 11
- receive() (*tmtccmd.com.dummy.DummyComIF* method), 19
- receive() (*tmtccmd.com.qemu.QEMUComIF* method), 21
- receive() (*tmtccmd.com.serial\_cobs.SerialCobsComIF* method), 15
- receive() (*tmtccmd.com.serial\_dle.SerialDleComIF* method), 16
- receive() (*tmtccmd.com.tcp.TcpSpacepacketsClient* method), 12
- receive() (*tmtccmd.com.udp.UdpClient* method), 14
- receive\_reply\_package() (*tmtccmd.com.dummy.DummyHandler* method), 19
- ReceptionDecodeError, 11
- RECV\_ADDRESS (*tmtccmd.com.tcpip\_utils.TcpIpConfigIds* attribute), 13
- RECV\_MAX\_SIZE (*tmtccmd.com.tcpip\_utils.TcpIpConfigIds* attribute), 13
- RED (*tmtccmd.util.conf\_util.AnsiColors* attribute), 67
- reduced\_printout (*tmtccmd.config.args.AppParams* attribute), 52
- register\_keyboard\_interrupt\_handler() (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend* method), 40
- RegularTmtcLogWrapper (class in *tmtccmd.logging.pus*), 62
- reporter\_id (*tmtccmd.pus.tm.s5\_fsfw\_event.EventDefinition* attribute), 28
- request (*tmtccmd.cfdp.request.CfdpRequestWrapper* property), 38
- request (*tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend* property), 40
- request\_type (*tmtccmd.cfdp.request.CfdpRequestWrapper* property), 38
- request\_type\_from\_args() (*tmtccmd.config.args.PostArgsParsingWrapper* method), 53
- RESET (*tmtccmd.util.conf\_util.AnsiColors* attribute), 67
- resume() (*tmtccmd.tmtc.ccsds\_seq\_sender.SequentialCcsdsSender* method), 50
- root\_node() (*tmtccmd.config.tmtc.CmdTreeNode* class method), 57
- rows (*tmtccmd.pus.s20\_fsfw\_param\_defs.FsfwParamId* attribute), 31
- rows (*tmtccmd.pus.s20\_fsfw\_param\_defs.Parameter* property), 32
- ## S
- save\_to\_json\_with\_prompt() (in module *tmtccmd.util.json*), 63
- SELECT\_ALL (*tmtccmd.pus.s11\_tc\_sched\_defs.TypeOfTimeWindow* attribute), 30
- send() (*tmtccmd.com.ComInterface* method), 11
- send() (*tmtccmd.com.dummy.DummyComIF* method), 19
- send() (*tmtccmd.com.qemu.QEMUComIF* method), 21



send() (*tmtccmd.com.serial\_cobs.SerialCobsComIF* method), 15  
 send() (*tmtccmd.com.serial\_dle.SerialDleComIF* method), 17  
 send() (*tmtccmd.com.tcp.TcpSpacepacketsClient* method), 12  
 send() (*tmtccmd.com.udp.UdpClient* method), 14  
 SEND\_ADDRESS (*tmtccmd.com.tcpip\_utils.TcpIpConfigIds* attribute), 13  
 send\_cb() (*tmtccmd.tmtc.handler.TcHandlerBase* method), 47  
 send\_data() (*tmtccmd.com.qemu.QEMUComIF* method), 21  
 send\_data\_async() (*tmtccmd.com.qemu.QEMUComIF* method), 21  
 SendCbParams (class in *tmtccmd.tmtc.handler*), 46  
 SenderMode (class in *tmtccmd.tmtc.ccsds\_seq\_sender*), 50  
 SendError, 11  
 SeqResultWrapper (class in *tmtccmd.tmtc.ccsds\_seq\_sender*), 50  
 SequentialCcsdsSender (class in *tmtccmd.tmtc.ccsds\_seq\_sender*), 50  
 SERIAL\_BAUD\_RATE (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SERIAL\_BAUDRATE (*tmtccmd.util.json.JsonKeyNames* attribute), 63  
 SERIAL\_COBS (*tmtccmd.config.defs.CoreComInterfaces* attribute), 60  
 SERIAL\_COMM\_TYPE (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SERIAL\_DLE (*tmtccmd.config.defs.CoreComInterfaces* attribute), 60  
 SERIAL\_DLE\_MAX\_FRAME\_SIZE (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SERIAL\_DLE\_QUEUE\_LEN (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SERIAL\_FRAME\_SIZE (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SERIAL\_HINT (*tmtccmd.util.json.JsonKeyNames* attribute), 63  
 serial\_port (*tmtccmd.com.serial\_base.SerialCfg* attribute), 17  
 SERIAL\_PORT (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SERIAL\_PORT (*tmtccmd.util.json.JsonKeyNames* attribute), 63  
 SERIAL\_QEMU (*tmtccmd.config.defs.CoreComInterfaces* attribute), 60  
 serial\_timeout (*tmtccmd.com.serial\_base.SerialCfg* attribute), 17  
 SERIAL\_TIMEOUT (*tmtccmd.com.serial\_base.SerialConfigIds* attribute), 18  
 SerialCfg (class in *tmtccmd.com.serial\_base*), 17  
 SerialCfgWrapper (class in *tmtccmd.config.com*), 59  
 SerialCobsComIF (class in *tmtccmd.com.serial\_cobs*), 15  
 SerialComBase (class in *tmtccmd.com.serial\_base*), 17  
 SerialCommunicationType (class in *tmtccmd.com.serial\_base*), 17  
 SerialConfigIds (class in *tmtccmd.com.serial\_base*), 17  
 SerialDleComIF (class in *tmtccmd.com.serial\_dle*), 16  
 service (*tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm* property), 35  
 service (*tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm* property), 29  
 service (*tmtccmd.tmtc.tm\_base.PusTmBase* property), 44  
 service (*tmtccmd.tmtc.tm\_base.PusTmInterface* property), 45  
 Service1FsfwWrapper (class in *tmtccmd.pus.tm.s1\_verification*), 25  
 Service200FsfwReader (class in *tmtccmd.pus.tm.s200\_fsfw\_mode*), 36  
 Service200FsfwTm (class in *tmtccmd.pus.tm.s200\_fsfw\_mode*), 37  
 Service20FsfwTm (class in *tmtccmd.pus.tm.s20\_fsfw\_param*), 35  
 Service20ParamDumpWrapper (class in *tmtccmd.pus.tm.s20\_fsfw\_param*), 35  
 Service3Base (class in *tmtccmd.pus.tm.s3\_hk\_base*), 25  
 Service5Tm (class in *tmtccmd.pus.tm.s5\_fsfw\_event*), 28  
 SERVICE\_200\_MODE (*tmtccmd.pus.CustomFsfwPusService* attribute), 24  
 set\_cfdp\_params\_with\_prompts() (*tmtccmd.config.args.PostArgsParsingWrapper* method), 53  
 set\_cfdp\_params\_without\_prompts() (*tmtccmd.config.args.PostArgsParsingWrapper* method), 53  
 set\_custom\_printout() (*tmtccmd.tmtc.tm\_base.PusTmInfoBase* method), 45  
 set\_dle\_settings() (*tmtccmd.com.qemu.QEMUComIF* method), 21  
 set\_fixed\_frame\_settings() (*tmtccmd.com.qemu.QEMUComIF* method), 21  
 set\_id (*tmtccmd.pus.tm.s3\_hk\_base.Service3Base* property), 25  
 set\_layer\_is\_last\_child() (*tmtccmd.com.qemu.QEMUComIF* method), 21

`cmd.config.tmtc.DepthInfo` method), 57  
`set_lock_timeout()` (in module `tmtc-  
cmd.core.globals_manager`), 42  
`set_packet_info()` (`tmtc-  
cmd.tmtc.tm_base.PusTmInfoBase` method), 45  
`set_packet_info()` (`tmtc-  
cmd.tmtc.tm_base.PusTmInfoInterface` method), 45  
`set_params()` (`tmtccmd.config.args.ProcedureParamsWrapper` method), 54  
`set_params_with_prompts()` (`tmtc-  
cmd.config.args.PostArgsParsingWrapper` method), 53  
`set_params_without_prompts()` (`tmtc-  
cmd.config.args.PostArgsParsingWrapper` method), 53  
`set_tmtc_params_with_prompts()` (`tmtc-  
cmd.config.args.PostArgsParsingWrapper` method), 53  
`set_tmtc_params_without_prompts()` (`tmtc-  
cmd.config.args.PostArgsParsingWrapper` method), 53  
`setup()` (in module `tmtccmd`), 39  
`setup_backend_def_procedure()` (in module `tmtc-  
cmd`), 39  
`SetupParams` (class in `tmtccmd.config.args`), 54  
`SetupWrapper` (class in `tmtccmd.config`), 51  
`Severity` (class in `tmtccmd.pus.s5_fsfw_event_defs`), 28  
`severity` (`tmtccmd.pus.s5_fsfw_event_defs.EventInfo` attribute), 28  
`severity` (`tmtccmd.pus.s5_satrs_event_defs.EventU32` attribute), 27  
`severity` (`tmtccmd.pus.tm.s5_fsfw_event.Service5Tm` property), 29  
`SHORT` (`tmtccmd.fsfw.tmtc_printer.DisplayMode` attribute), 66  
`sort()` (`tmtccmd.config.tmtc.TmtcDefinitionWrapper` method), 58  
`sort_num_key_dict()` (`tmtc-  
cmd.config.tmtc.OpCodeEntry` method), 57  
`sort_text_key_dict()` (`tmtc-  
cmd.config.tmtc.OpCodeEntry` method), 57  
`source_data` (`tmtccmd.pus.tm.s20_fsfw_param.Service20FsfwTm` property), 35  
`source_data` (`tmtccmd.pus.tm.s5_fsfw_event.Service5Tm` property), 29  
`source_file` (`tmtccmd.config.defs.CfdpParams` attribute), 60  
`sp_header` (`tmtccmd.pus.tm.s20_fsfw_param.Service20FsfwTm` property), 35  
`sp_header` (`tmtccmd.pus.tm.s5_fsfw_event.Service5Tm` property), 29  
`SPACE_PACKET_ID` (`tmtc-  
cmd.com.tcpip_utils.TcpIpConfigIds` attribute), 13  
`SPACE_PACKETS` (`tmtc-  
cmd.com.tcp.TcpCommunicationType` attribute), 11  
`SpacePacketEntry` (class in `tmtccmd.tmtc.queue`), 48  
`SpecificApidHandlerBase` (class in `tmtc-  
cmd.tmtc.common`), 44  
`ssc` (`tmtccmd.tmtc.tm_base.PusTmBase` property), 44  
`ssc` (`tmtccmd.tmtc.tm_base.PusTmInterface` property), 46  
`start()` (in module `tmtccmd`), 39  
`start()` (`tmtccmd.core.base.FrontendBase` method), 42  
`start()` (`tmtccmd.core.ccsds_backend.CcsdsTmtcBackend` method), 40  
`start_background_loop()` (in module `tmtc-  
cmd.com.qemu`), 23  
`start_dle_polling()` (`tmtc-  
cmd.com.qemu.QEMUComIF` method), 21  
`state` (`tmtccmd.core.ccsds_backend.CcsdsTmtcBackend` property), 41  
`step_num()` (`tmtccmd.pus.VerificationWrapper` static method), 24  
`stop()` (`tmtccmd.com.qemu.QmpConnection` method), 22  
`store_com_if_json()` (in module `tmtccmd.com.utils`), 20  
`str_for_tree()` (`tmtccmd.config.tmtc.CmdTreeNode` method), 57  
`str_to_severity()` (in module `tmtc-  
cmd.pus.s5_fsfw_event_defs`), 28  
`Subservice` (class in `tmtccmd.pus.s11_tc_sched_defs`), 29  
`Subservice` (class in `tmtc-  
cmd.pus.s200_fsfw_mode_defs`), 36  
`Subservice` (class in `tmtc-  
cmd.pus.s201_fsfw_health_defs`), 37  
`subservice` (`tmtccmd.pus.tm.s20_fsfw_param.Service20FsfwTm` property), 35  
`subservice` (`tmtccmd.pus.tm.s5_fsfw_event.Service5Tm` property), 29  
`subservice` (`tmtccmd.tmtc.tm_base.PusTmBase` property), 44  
`subservice` (`tmtccmd.tmtc.tm_base.PusTmInterface` property), 46

## T

`TC_ANNOUNCE_HEALTH` (`tmtc-  
cmd.pus.s201_fsfw_health_defs.Subservice` attribute), 37  
`TC_ANNOUNCE_HEALTH_ALL` (`tmtc-  
cmd.pus.s201_fsfw_health_defs.Subservice`

attribute), 37  
 TC\_DELETE (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 29  
 TC\_DELETE\_WITH\_FILTER (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 29  
 TC\_DETAIL\_REPORT\_FILTER\_BASED (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_DETAIL\_REPORT\_TIME\_BASED (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_DISABLE (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_DUMP (tmtccmd.pus.s20\_fsfw\_param\_defs.CustomSubservice attribute), 31  
 TC\_ENABLE (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_FUNCTIONAL\_CMD (tmtccmd.pus.s8\_fsfw\_action\_defs.CustomSubservice attribute), 29  
 TC\_GEN\_EVENT (tmtccmd.pus.s17\_test\_defs.CustomSubservice attribute), 30  
 TC\_INSERT (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_LOAD (tmtccmd.pus.s20\_fsfw\_param\_defs.CustomSubservice attribute), 31  
 tc\_mode (tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend property), 41  
 TC\_MODE\_ANNOUNCE (tmtccmd.pus.s200\_fsfw\_mode\_defs.Subservice attribute), 36  
 TC\_MODE\_ANNOUNCE\_RECURSIVE (tmtccmd.pus.s200\_fsfw\_mode\_defs.Subservice attribute), 36  
 TC\_MODE\_COMMAND (tmtccmd.pus.s200\_fsfw\_mode\_defs.Subservice attribute), 36  
 TC\_MODE\_COMMAND\_FORCES (tmtccmd.pus.s200\_fsfw\_mode\_defs.Subservice attribute), 36  
 TC\_MODE\_READ (tmtccmd.pus.s200\_fsfw\_mode\_defs.Subservice attribute), 36  
 tc\_operation() (tmtccmd.core.ccsds\_backend.CcsdsTmtcBackend method), 41  
 tc\_prefix() (tmtccmd.logging.pus.RawTmtcLogBase static method), 62  
 TC\_RESET (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_SET\_HEALTH (tmtccmd.pus.s201\_fsfw\_health\_defs.Subservice attribute), 37  
 TC\_TIMESHIFT (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_TIMESHIFT\_ALL (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TC\_TIMESHIFT\_WITH\_FILTER (tmtccmd.pus.s11\_tc\_sched\_defs.Subservice attribute), 30  
 TcHandlerBase (class in tmtccmd.tmtc.handler), 46  
 TcMode (class in tmtccmd.core.base), 42  
 TCP (tmtccmd.com.tcpip\_utils.TcpIpType attribute), 13  
 TCP (tmtccmd.config.defs.CoreComInterfaces attribute), 61  
 TcpCommunicationType (class in tmtccmd.com.tcp), 11  
 TCPIP\_TCP\_DEST\_IP\_ADDRESS (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_TCP\_DEST\_PORT (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_TCP\_RECV\_MAX\_SIZE (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_UDP\_DEST\_IP\_ADDRESS (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_UDP\_DEST\_PORT (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_UDP\_RECV\_IP\_ADDRESS (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_UDP\_RECV\_MAX\_SIZE (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TCPIP\_UDP\_RECV\_PORT (tmtccmd.util.json.JsonKeyNames attribute), 63  
 TcpipCfg (class in tmtccmd.config.com), 59  
 TcpIpConfigIds (class in tmtccmd.com.tcpip\_utils), 12  
 TcpIpType (class in tmtccmd.com.tcpip\_utils), 13  
 TcProcedureBase (class in tmtccmd.tmtc.procedure), 49  
 TcProcedureType (class in tmtccmd.tmtc.procedure), 49  
 TcpSpacepacketsClient (class in tmtccmd.com.tcp), 11  
 TcQueueEntryBase (class in tmtccmd.tmtc.queue), 48  
 TcQueueEntryType (class in tmtccmd.tmtc.queue), 49  
 TcSchedReqId (class in tmtccmd.pus.s11\_tc\_sched\_defs), 30  
 TERMINATION\_NO\_ERROR (tmtccmd.core.base.BackendRequest attribute), 42  
 tick\_mark\_unicode() (in module tmtccmd.pus), 25  
 TimedLogWhen (class in tmtccmd.logging.pus), 62  
 timestamp (tmtccmd.pus.tm.s20\_fsfw\_param.Service20FsfwTm property), 35  
 timestamp (tmtccmd.pus.tm.s5\_fsfw\_event.Service5Tm property), 29  
 TM\_CANT\_REACH\_MODE (tmtccmd.pus.s200\_fsfw\_mode\_defs.Subservice attribute), 36  
 tm\_data (tmtccmd.tmtc.tm\_base.PusTmBase property), 44

<code>tm_data</code> ( <i>tmtccmd.tmtc.tm_base.PusTmInterface</i> property), 46	<code>tmtccmd.com.serial_dle</code> module, 16
<code>TM_DATA_REPLY</code> ( <i>tmtccmd.pus.s8_fsw_action_defs.CustomSubservice</i> attribute), 29	<code>tmtccmd.com.tcp</code> module, 11
<code>TM_DETAIL_REPORT_FILTER_BASED</code> ( <i>tmtccmd.pus.s11_tc_sched_defs.Subservice</i> attribute), 30	<code>tmtccmd.com.tcpip_utils</code> module, 12
<code>TM_DETAIL_REPORT_TIME_BASED</code> ( <i>tmtccmd.pus.s11_tc_sched_defs.Subservice</i> attribute), 30	<code>tmtccmd.com.udp</code> module, 13
<code>TM_DUMP_REPLY</code> ( <i>tmtccmd.pus.s20_fsw_param_defs.CustomSubservice</i> attribute), 31	<code>tmtccmd.com.utils</code> module, 20
<code>TM_HEALTH_SET</code> ( <i>tmtccmd.pus.s201_fsw_health_defs.Subservice</i> attribute), 37	<code>tmtccmd.config</code> module, 51
<code>tm_listener</code> ( <i>tmtccmd.core.ccsds_backend.CcsdsTmtcBackend</i> module property), 41	<code>tmtccmd.config.args</code> module, 52
<code>tm_mode</code> ( <i>tmtccmd.core.ccsds_backend.CcsdsTmtcBackend</i> module property), 41	<code>tmtccmd.config.cfdp</code> module, 58
<code>TM_MODE_REPLY</code> ( <i>tmtccmd.pus.s200_fsw_mode_defs.Subservice</i> attribute), 36	<code>tmtccmd.config.com</code> module, 59
<code>tm_operation()</code> ( <i>tmtccmd.core.ccsds_backend.CcsdsTmtcBackend</i> method), 41	<code>tmtccmd.config.defs</code> module, 60
<code>tm_prefix()</code> ( <i>tmtccmd.logging.pus.RawTmtcLogBase</i> static method), 62	<code>tmtccmd.config.hook</code> module, 51
<code>TM_WRONG_MODE_REPLY</code> ( <i>tmtccmd.pus.s200_fsw_mode_defs.Subservice</i> attribute), 36	<code>tmtccmd.config.objects</code> module, 61
<code>TmHandlerBase</code> (class in <i>tmtccmd.tmtc.common</i> ), 44	<code>tmtccmd.config.tmtc</code> module, 56
<code>TmMode</code> (class in <i>tmtccmd.core.base</i> ), 42	<code>tmtccmd.core</code> module, 40
<code>tmtc_definitions_provider()</code> (in module <i>tmtccmd.config.tmtc</i> ), 58	<code>tmtccmd.core.base</code> module, 41
<code>tmtc_params_to_procedure()</code> (in module <i>tmtccmd.config</i> ), 51	<code>tmtccmd.core.ccsds_backend</code> module, 40
<code>tmtccmd</code> module, 38	<code>tmtccmd.core.globals_manager</code> module, 42
<code>tmtccmd.cfdp.request</code> module, 38	<code>tmtccmd.fsw</code> module, 68
<code>tmtccmd.com</code> module, 10	<code>tmtccmd.fsw.tmtc_printer</code> module, 66
<code>tmtccmd.com.dummy</code> module, 18	<code>tmtccmd.logging.pus</code> module, 62
<code>tmtccmd.com.qemu</code> module, 20	<code>tmtccmd.pus</code> module, 24
<code>tmtccmd.com.ser_utils</code> module, 18	<code>tmtccmd.pus.s11_tc_sched</code> module, 29
<code>tmtccmd.com.serial_base</code> module, 17	<code>tmtccmd.pus.s11_tc_sched_defs</code> module, 29
<code>tmtccmd.com.serial_cobs</code> module, 15	<code>tmtccmd.pus.s17_test</code> module, 30
	<code>tmtccmd.pus.s17_test_defs</code> module, 30
	<code>tmtccmd.pus.s1_verification</code> module, 25
	<code>tmtccmd.pus.s200_fsw_mode</code> module, 36

---

tmtccmd.pus.s200_fsfw_mode_defs	tmtccmd.tmtc.handler
module, 36	module, 46
tmtccmd.pus.s201_fsfw_health	tmtccmd.tmtc.procedure
module, 37	module, 49
tmtccmd.pus.s201_fsfw_health_defs	tmtccmd.tmtc.queue
module, 37	module, 47
tmtccmd.pus.s20_fsfw_param	tmtccmd.tmtc.tm_base
module, 31	module, 44
tmtccmd.pus.s20_fsfw_param_defs	tmtccmd.util
module, 31	module, 68
tmtccmd.pus.s5_fsfw_event	tmtccmd.util.conf_util
module, 28	module, 67
tmtccmd.pus.s5_fsfw_event_defs	tmtccmd.util.exit
module, 28	module, 65
tmtccmd.pus.s5_satrs_event	tmtccmd.util.hammingcode
module, 27	module, 65
tmtccmd.pus.s5_satrs_event_defs	tmtccmd.util.json
module, 27	module, 63
tmtccmd.pus.s8_fsfw_action	tmtccmd.util.obj_id
module, 29	module, 64
tmtccmd.pus.s8_fsfw_action_defs	tmtccmd.version
module, 29	module, 63
tmtccmd.pus.tc.s200_fsfw_mode	TmtcDefinitionWrapper (class in tmtccmd.config.tmtc), 57
module, 36	TmTypes (class in tmtccmd.tmtc.common), 44
tmtccmd.pus.tc.s20_fsfw_param	to_cfdp_procedure() (tmtccmd.tmtc.procedure.ProcedureWrapper method), 49
module, 33	to_custom_procedure() (tmtccmd.tmtc.procedure.ProcedureWrapper method), 49
tmtccmd.pus.tc.s3_fsfw_hk	to_log_entry() (tmtccmd.tmtc.queue.QueueEntryHelper method), 48
module, 25	to_packet_delay_entry() (tmtccmd.tmtc.queue.QueueEntryHelper method), 48
tmtccmd.pus.tc.s5_event	to_pus_tc_entry() (tmtccmd.tmtc.queue.QueueEntryHelper method), 48
module, 27	to_put_request() (tmtccmd.cfdp.request.CfdpRequestWrapper method), 38
tmtccmd.pus.tc.s8_fsfw_action	to_raw_tc_entry() (tmtccmd.tmtc.queue.QueueEntryHelper method), 48
module, 29	to_space_packet_entry() (tmtccmd.tmtc.queue.QueueEntryHelper method), 48
tmtccmd.pus.tm.s1_verification	TO_TIMETAG (tmtccmd.pus.s11_tc_sched_defs.TypeOfTimeWindow attribute), 30
module, 25	to_tree_commanding_procedure() (tmtccmd.tmtc.procedure.ProcedureWrapper method), 49
tmtccmd.pus.tm.s200_fsfw_mode	
module, 36	
tmtccmd.pus.tm.s20_fsfw_param	
module, 35	
tmtccmd.pus.tm.s2_rawcmd	
module, 25	
tmtccmd.pus.tm.s3_fsfw_hk	
module, 25	
tmtccmd.pus.tm.s3_hk_base	
module, 25	
tmtccmd.pus.tm.s5_fsfw_event	
module, 28	
tmtccmd.pus.tm.s8_fsfw_action	
module, 29	
tmtccmd.tmtc.ccsds_seq_sender	
module, 50	
tmtccmd.tmtc.ccsds_tm_listener	
module, 43	
tmtccmd.tmtc.common	
module, 43	



[to\\_tuple \(tmtccmd.com.tcpip\\_utils.EthAddr property\), 12](#)  
[to\\_wait\\_entry\(\) \(tmtccmd.tmtc.queue.QueueEntryHelper method\), 48](#)  
[transmission\\_mode \(tmtccmd.config.defs.CfdpParams attribute\), 60](#)  
[TREE\\_COMMANDING \(tmtccmd.tmtc.procedure.TcProcedureType attribute\), 50](#)  
[tree\\_commanding\\_params\(\) \(tmtccmd.config.args.ProcedureParamsWrapper method\), 54](#)  
[tree\\_print\\_max\\_depth \(tmtccmd.config.args.CommandingParams attribute\), 53](#)  
[tree\\_print\\_with\\_description \(tmtccmd.config.args.CommandingParams attribute\), 53](#)  
[TreeCommandingParams \(class in tmtccmd.config.defs\), 61](#)  
[TreeCommandingProcedure \(class in tmtccmd.tmtc.procedure\), 50](#)  
[TreePart \(class in tmtccmd.config.tmtc\), 58](#)  
[try\\_set\\_com\\_if\(\) \(tmtccmd.core.ccsds\\_backend.CcsdsTmtcBackend method\), 41](#)  
[TypeOfTimeWindow \(class in tmtccmd.pus.s11\\_tc\\_sched\\_defs\), 30](#)

## U

[UDP \(tmtccmd.com.tcpip\\_utils.TcpIpType attribute\), 13](#)  
[UDP \(tmtccmd.config.defs.CoreComInterfaces attribute\), 61](#)  
[UDP\\_RECV \(tmtccmd.com.tcpip\\_utils.TcpIpType attribute\), 13](#)  
[UdpClient \(class in tmtccmd.com.udp\), 13](#)  
[unique\\_id \(tmtccmd.pus.s20\\_fsfw\\_param\\_defs.ParameterId attribute\), 32](#)  
[unique\\_id \(tmtccmd.pus.s5\\_satrs\\_event\\_defs.EventU32 attribute\), 27](#)  
[unlock\\_global\\_pool\(\) \(in module tmtccmd.core.globals\\_manager\), 42](#)  
[unpack\(\) \(tmtccmd.pus.s20\\_fsfw\\_param\\_defs.FsfwParamId class method\), 31](#)  
[unpack\(\) \(tmtccmd.pus.s20\\_fsfw\\_param\\_defs.Parameter class method\), 32](#)  
[unpack\(\) \(tmtccmd.pus.s20\\_fsfw\\_param\\_defs.ParameterId class method\), 32](#)  
[unpack\(\) \(tmtccmd.pus.s5\\_satrs\\_event\\_defs.EventU32 class method\), 27](#)  
[unpack\(\) \(tmtccmd.pus.tm.s200\\_fsfw\\_mode.Service200FsfwTm class method\), 37](#)  
[unpack\(\) \(tmtccmd.pus.tm.s20\\_fsfw\\_param.Service20FsfwTm class method\), 35](#)  
[unpack\(\) \(tmtccmd.pus.tm.s5\\_fsfw\\_event.Service5Tm class method\), 29](#)  
[UNSPECIFIED \(tmtccmd.config.defs.CoreComInterfaces attribute\), 61](#)  
[update\\_global\(\) \(in module tmtccmd.core.globals\\_manager\), 42](#)  
[Usart \(class in tmtccmd.com.qemu\), 22](#)  
[UsartProtocol \(class in tmtccmd.com.qemu\), 23](#)  
[UsartStatusException, 23](#)  
[use\\_ansi\\_colors \(tmtccmd.config.args.AppParams attribute\), 52](#)  
[use\\_gui \(tmtccmd.config.args.AppParams attribute\), 52](#)  
[use\\_gui \(tmtccmd.config.args.PostArgsParsingWrapper property\), 53](#)  
[use\\_gui \(tmtccmd.config.args.SetupParams property\), 54](#)  
[user\\_hook\(\) \(tmtccmd.tmtc.common.CcsdsTmHandler method\), 44](#)

## V

[valid \(tmtccmd.tmtc.tm\\_base.PusTmInterface property\), 46](#)  
[validity\\_buffer\\_list\(\) \(in module tmtccmd.fsfw\), 68](#)  
[VerificationWrapper \(class in tmtccmd.pus\), 24](#)  
[verificator \(tmtccmd.pus.VerificationWrapper property\), 24](#)

## W

[WAIT \(tmtccmd.tmtc.queue.TcQueueEntryType attribute\), 49](#)  
[WaitEntry \(class in tmtccmd.tmtc.queue\), 49](#)  
[wrapped\\_prompt\(\) \(in module tmtccmd.util.conf\\_util\), 68](#)  
[write\(\) \(tmtccmd.com.qemu.Usart method\), 23](#)

## Y

[YELLOW \(tmtccmd.util.conf\\_util.AnsiColors attribute\), 67](#)