
Yamcs Python Client Documentation

Release 1.7.5

Space Applications Services

Oct 31, 2021

CONTENTS

1	General Client	3
1.1	Reference	3
1.2	Snippets	13
2	Mission Database	15
2.1	Reference	16
2.2	Snippets	21
3	TM/TC Processing	23
3.1	Reference	23
3.2	Snippets	37
4	Archive	41
4.1	Reference	41
4.2	Snippets	49
5	Link Management	53
5.1	Reference	53
5.2	Snippets	55
6	Object Storage	57
6.1	Reference	57
6.2	Snippets	60
7	File Transfer	61
7.1	Reference	61
7.2	Snippets	63
8	Time Correlation (TCO)	65
8.1	Reference	65
8.2	Snippets	67
9	Examples	69
9.1	alarms.py	69
9.2	archive_breakdown.py	70
9.3	archive_retrieval.py	71
9.4	authenticate.py	73
9.5	ccsds_completeness.py	74
9.6	commanding.py	75
9.7	cop1.py	77
9.8	file_transfer.py	78

9.9	links.py	79
9.10	events.py	79
9.11	mission_time.py	80
9.12	packet_subscription.py	80
9.13	parameter_subscription.py	81
9.14	plot_with_matplotlib.py	82
9.15	query_mdb.py	83
9.16	read_write_parameters.py	84
Python Module Index		87
Index		89

Getting Started

Install Python 3.6 or higher.

Install `yamcs-client` from PyPI:

```
pip install --upgrade yamcs-client
```

Usage

Get domain-specific clients:

```
from yamcs.client import YamcsClient
client = YamcsClient('localhost:8090')

mdb = client.get_mdb(instance='simulator')
# ...

archive = client.get_archive(instance='simulator')
# ...

processor = client.get_processor(instance='simulator', processor='realtime')
# ...
```

Documentation

- *General Client*
- *Mission Database*
- *TM/TC Processing*
- *Archive*
- *Link Management*
- *Object Storage*
- *File Transfer*
- *Time Correlation (TCO)*

Examples

- *Examples*

GENERAL CLIENT

1.1 Reference

1.1.1 YamcsClient

class `yamcs.client.YamcsClient` (*address*, ***kwargs*)

Bases: `object`

Client for accessing core Yamcs resources.

Parameters

- **address** (*str*) – The address of Yamcs in the format ‘hostname:port’
- **tls** (*Optional[bool]*) – Whether TLS encryption is expected
- **tls_verify** (*Optional[Union[bool, str]]*) – Whether server certificate verification is enabled (only applicable if `tls=True`). As an alternative to a boolean value, this option may be set to a path containing the appropriate TLS CA certificate bundle.
- **credentials** (*Optional[Credentials]*) – Credentials for when the server is secured
- **user_agent** (*Optional[str]*) – Optionally override the default user agent

create_event_subscription (*instance*, *on_data*, *timeout=60*)

Create a new subscription for receiving events of an instance.

This method returns a future, then returns immediately. Stop the subscription by canceling the future.

Parameters

- **instance** (*str*) – A Yamcs instance name
- **on_data** (*Optional[Callable[Event]]*) – Function that gets called on each `Event`.
- **timeout** (*Optional[float]*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription.

Return type `WebSocketSubscriptionFuture`

create_instance (*name*, *template*, *args=None*, *labels=None*)

Create a new instance based on an existing template. This method blocks until the instance is fully started.

Parameters

- **instance** (*str*) – A Yamcs instance name.

- **template** (*str*) – The name of an existing template.

create_link_subscription (*instance*, *on_data=None*, *timeout=60*)

Create a new subscription for receiving data link updates of an instance.

This method returns a future, then returns immediately. Stop the subscription by canceling the future.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **on_data** (*Optional[Callable[LinkEvent]]*) – Function that gets called with *LinkEvent* updates.
- **timeout** (*Optional[float]*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription.

Return type *LinkSubscription*

create_time_subscription (*instance*, *on_data=None*, *timeout=60*)

Create a new subscription for receiving time updates of an instance. Time updates are emitted at 1Hz.

This method returns a future, then returns immediately. Stop the subscription by canceling the future.

Parameters

- **instance** (*str*) – A Yamcs instance name
- **on_data** (*Optional[Callable[datetime]]*) – Function that gets called with *datetime* updates.
- **timeout** (*Optional[float]*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription.

Return type *TimeSubscription*

get_archive (*instance*)

Return an object for working with the Archive of the specified instance.

Parameters **instance** (*str*) – A Yamcs instance name.

Return type *ArchiveClient*

get_auth_info ()

Returns general authentication information. This operation does not require authenticating and is useful to test if a server requires authentication or not.

Return type *AuthInfo*

get_file_transfer_client (*instance*)

Return an object for working with file transfers on a specified instance.

Parameters **instance** (*str*) – A Yamcs instance name.

Return type *FileTransferClient*

get_link (*instance*, *link*)

Return an object for working with a specific Yamcs link.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **link** (*str*) – A link name within that instance.

Return type *LinkClient*

get_mdb (*instance*)

Return an object for working with the MDB of the specified instance.

Parameters **instance** (*str*) – A Yamcs instance name.

Return type *MDBClient*

get_processor (*instance, processor*)

Return an object for working with a specific Yamcs processor.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **processor** (*str*) – A processor name within that instance.

Return type *ProcessorClient*

get_server_info ()

Return general server info.

Return type *ServerInfo*

get_storage_client (*instance='_global'*)

Return an object for working with object storage

Parameters **instance** (*str*) – The storage instance.

Return type *StorageClient*

get_tco_client (*instance, service*)

Return an object for Time Correlation API calls on a specified service.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **service** (*str*) – Target service name.

Return type *TCOClient*

get_time (*instance*)

Return the current mission time for the specified instance.

Return type *datetime*

get_user_info ()

Get information on the authenticated user.

Return type *UserInfo*

list_data_links (*instance*)

Deprecated. Use `list_links(instance)`.

list_instance_templates ()

List the available instance templates.

list_instances ()

Lists the instances.

Instances are returned in lexicographical order.

Return type *Iterable[Instance]*

list_links (*instance*)

Lists the data links visible to this client.

Data links are returned in random order.

Parameters **instance** (*str*) – A Yamcs instance name.

Return type `Iterable[Link]`

list_processors (*instance=None*)

Lists the processors.

Processors are returned in lexicographical order.

Parameters **instance** (*Optional[str]*) – A Yamcs instance name.

Return type `Iterable[Processor]`

list_services (*instance*)

List the services for an instance.

Parameters **instance** (*str*) – A Yamcs instance name.

Return type `Iterable[Service]`

restart_instance (*instance*)

Restarts a single instance.

Parameters **instance** (*str*) – A Yamcs instance name.

send_event (*instance, message, event_type=None, time=None, severity='info', source=None, sequence_number=None*)

Post a new event.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **message** (*str*) – Event message.
- **event_type** (*Optional[str]*) – Type of event.
- **severity** (*Optional[str]*) – The severity level of the event. One of `info`, `watch`, `warning`, `critical` or `severe`. Defaults to `info`.
- **time** (*Optional[datetime]*) – Time of the event. If unspecified, defaults to mission time.
- **source** (*Optional[str]*) – Source of the event. Useful for grouping events in the archive. When unset this defaults to `User`.
- **sequence_number** (*Optional[int]*) – Sequence number of this event. This is primarily used to determine unicity of events coming from the same source. If not set Yamcs will automatically assign a sequential number as if every submitted event is unique.

start_instance (*instance*)

Starts a single instance.

Parameters **instance** (*str*) – A Yamcs instance name.

start_service (*instance, service*)

Starts a single service.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **service** (*str*) – The name of the service.

stop_instance (*instance*)

Stops a single instance.

Parameters **instance** (*str*) – A Yamcs instance name.

stop_service (*instance, service*)

Stops a single service.

Parameters

- **instance** (*str*) – A Yamcs instance name.
- **service** (*str*) – The name of the service.

class `yamcs.client.TimeSubscription` (*manager*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object providing access to time updates.

A subscription object stores the last time info.

Initializes the future. Should not be called by clients.

time = `None`

The last time info.

Type `datetime`

class `yamcs.client.LinkSubscription` (*manager*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object providing access to data link updates.

A subscription object stores the last link info for each link.

Initializes the future. Should not be called by clients.

get_link (*name*)

Returns the latest link state.

Parameters **name** (*str*) – Identifying name of the data link

Return type `Link`

list_links ()

Returns a snapshot of all instance links.

Return type `Link[]`

1.1.2 Model

class `yamcs.model.AuthInfo` (*proto*)

Bases: `object`

Authentication information

property `require_authentication`

class `yamcs.model.Event` (*proto*)

Bases: `object`

A timetagged free-text message. Events work a lot like log messages in logging frameworks, but then targeted at operators.

property `event_type`

The event type. This is mission-specific and can be any string.

property generation_time

The time when the event was generated.

Type `datetime`

property message

Event message.

property reception_time

The time when the event was received by Yamcs.

Type `datetime`

property sequence_number

Sequence number. Usually this is assigned by the source of the event.

property severity

Severity level of the event. One of INFO, WATCH, WARNING, DISTRESS, CRITICAL or SEVERE.

property source

The event source. Can be any string.

class `yamcs.model.Instance` (*proto*)

Bases: `object`

property failure_cause

Failure message when `state == 'FAILED'`

property mission_time

Mission time of this instance's time service.

Type `datetime`

property name

Name of this instance.

property state

State of this instance. One of OFFLINE, INITIALIZING, INITIALIZED, STARTING, RUNNING, STOPPING or FAILED.

class `yamcs.model.InstanceTemplate` (*proto*)

Bases: `object`

A template for creating an instance.

property name

Name of this template.

class `yamcs.model.Link` (*proto*)

Bases: `object`

Represents a link with an external system. Depending on the semantics of the link, this may imply inbound data, outbound data or a combination of both.

property class_name

Name of this link's class.

property enabled

If `True`, this link accepts or outputs data.

property in_count

The number of inbound data events (example: packet count).

property instance

Name of the instance where this link is defined.

property name

Name of this link (unique per instance).

property out_count

The number of outbound data events (example: command count).

property status

Short status.

class `yamcs.model.LinkEvent` (*proto*)

Bases: `object`

Data holder used in link subscriptions.

property event_type

The type of the event. One of REGISTERED, UNREGISTERED, or UPDATED.

property link

Link state at the time of this event.

Type `Link`

class `yamcs.model.ObjectPrivilege` (*proto*)

Bases: `object`

property name**property objects**

class `yamcs.model.Processor` (*proto*)

Bases: `object`

property instance

Name of the instance where this processor is defined.

property mission_time

Mission time of this processor.

Type `datetime`

property name

Name of this processor.

property owner

User that owns this processor.

property persistent

If `True`, this processor does not close if no clients are connected.

property state

State of this processor.

property type

Type of this processor.

class `yamcs.model.ServerInfo` (*proto*)

Bases: `object`

General server properties.

property default_yamcs_instance

Returns the default Yamcs instance.

property id

The Server ID.

property version

The version of Yamcs Server.

class `yamcs.model.Service` (*proto*)

Bases: `object`

A Yamcs service.

property class_name

Name of this service's class.

property instance

Name of the instance where this service is defined.

property name

Name of this service.

property processor

Name of the processor where this service is defined.

property state

State of this service.

class `yamcs.model.UserInfo` (*proto*)

Bases: `object`

Info on a Yamcs User.

property object_privileges

property superuser

property system_privileges

property username

1.1.3 Authentication

User Accounts

Yamcs Server can be configured for different authentication setups.

The common use case is to entrust Yamcs with validating user credentials (either by locally verifying passwords, or by delegating to an upstream server such as an LDAP tree).

To authenticate in such a scenario simply do:

```
credentials = Credentials(username="admin", password="password")
client = YamcsClient("localhost:8090", credentials=credentials)
```

In the background this will convert your username/password credentials to an access token with limited lifetime, and a long-lived refresh token for automatically generating new access tokens.

Further HTTP requests do not use your username/password but instead use these tokens.

Service Accounts

Service accounts are useful in server-to-server scenarios. Support for service accounts will be available in future releases.

Types

```
class yamcs.core.auth.Credentials(username=None, password=None, access_token=None,  
refresh_token=None, expiry=None, client_id=None,  
client_secret=None, become=None)
```

Bases: `object`

Data holder for different types of credentials. Currently this includes:

- Username/password credentials (fields `username` and `password`)
- Bearer tokens (fields `access_token` and optionally `refresh_token`)

access_token = None
Short-lived bearer token.

become = None
Name of the user to impersonate. Only service accounts with impersonation authority can use this feature.

before_request (*session, auth_url*)

client_id = None
The client ID. Currently used only by service accounts.

client_secret = None
The client secret. Currently used only by service accounts.

expiry = None
When this token expires.

is_expired ()

login (*session, auth_url, on_token_update*)

password = None
Clear-text password (consider TLS!).

refresh (*session, auth_url*)

refresh_token = None
Refresh token used to request a new access token.

username = None
Username (only needed when using username/password credentials).

1.1.4 Exceptions

exception yamcs.core.exceptions.ConnectionFailure
Bases: `yamcs.core.exceptions.YamcsError`

Yamcs is not or no longer available.

exception yamcs.core.exceptions.NotFound
Bases: `yamcs.core.exceptions.YamcsError`

The resource was not found.

exception `yamcs.core.exceptions.TimeoutError`

Bases: `yamcs.core.exceptions.YamcsError`

The operation exceeded the given deadline.

exception `yamcs.core.exceptions.Unauthorized`

Bases: `yamcs.core.exceptions.YamcsError`

Unable to get access the resource.

exception `yamcs.core.exceptions.YamcsError`

Bases: `Exception`

Base class for raised exceptions.

1.1.5 Futures

class `yamcs.core.futures.WebSocketSubscriptionFuture` (*manager*)

Bases: `concurrent.futures._base.Future`

Future for capturing the asynchronous execution of a WebSocket subscription.

Initializes the future. Should not be called by clients.

add_done_callback (*fn*)

Attaches a callable that will be called when the future finishes.

Args:

fn: A callable that will be called with this future as its only argument when the future completes or is cancelled. The callable will always be called by a thread in the same process in which it was added. If the future has already completed or been cancelled then the callable will be called immediately. These callables are called in the order that they were added.

cancel ()

Closes the websocket and shutdowns the background thread consuming messages.

cancelled ()

Return True if the future was cancelled.

done ()

Return True if the future was cancelled or finished executing.

exception (*timeout=None*)

Return the exception raised by the call that the future represents.

Args:

timeout: The number of seconds to wait for the exception if the future isn't done. If None, then there is no limit on the wait time.

Returns: The exception raised by the call that the future represents or None if the call completed without raising.

Raises: `CancelledError`: If the future was cancelled. `TimeoutError`: If the future didn't finish executing before the given

`timeout`.

reply (*timeout=None*)

Returns the initial reply. This is emitted before any subscription data is emitted. This function raises an exception if the subscription attempt failed.

result (*timeout=None*)

Return the result of the call that the future represents.

Args:

timeout: The number of seconds to wait for the result if the future isn't done. If None, then there is no limit on the wait time.

Returns: The result of the call that the future represents.

Raises: CanceledError: If the future was cancelled. TimeoutError: If the future didn't finish executing before the given

timeout.

Exception: If the call raised then that exception will be raised.

running ()

Return True if the future is currently executing.

set_exception (*exception*)

Sets the result of the future as being the given exception.

Should only be used by Executor implementations and unit tests.

set_result (*result*)

Sets the return value of work associated with the future.

Should only be used by Executor implementations and unit tests.

1.2 Snippets

Create a *YamcsClient*:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
```

Provide credentials if Yamcs is secured:

```
from yamcs.client import YamcsClient
from yamcs.core.auth import Credentials

credentials = Credentials(username='admin', password='password')
client = YamcsClient('localhost:8090', credentials=credentials)
```

1.2.1 Events

Receive *Event* callbacks:

```
def callback(event):
    print("Event:", event)

client.create_event_subscription(instance="simulator", on_data=callback)

sleep(5) # Subscription is non-blocking
```

Send an event:

```
client.send_event(instance="simulator", message="hello world")
```

MISSION DATABASE

The Mission Database API provides methods for reading the entries in a Yamcs Mission Database (MDB). An MDB groups telemetry and command definitions for one or more *space systems*. The MDB is used to:

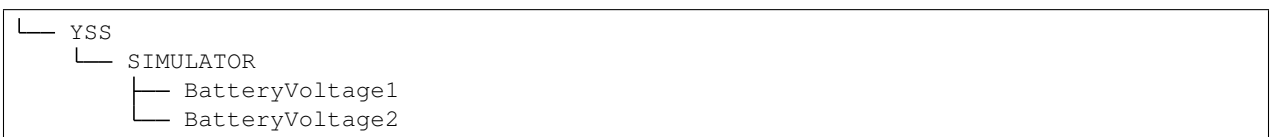
- Instruct Yamcs how to process incoming packets
- Describe items in Yamcs Archive
- Instruct Yamcs how to compose telecommands

Space systems form a hierarchical multi-rooted tree. Each level of the tree may contain any number of definitions. These break down in:

- parameters
- containers
- commands
- algorithms

Entries in the Space system are addressable via a qualified name that looks like a Unix file path. Each segment of the path contains the name of the space system node, the final path segment is the name of the entry itself.

For example, in an MDB that contains these parameter entries:



we find two space systems `/YSS` and `/YSS/SIMULATOR` and two parameter entries `/YSS/SIMULATOR/BatteryVoltage1` and `/YSS/SIMULATOR/BatteryVoltage2`.

Some MDB entries may also define an alias. An alias is a unique name to address the entry under a custom namespace (unrelated to XTCE space systems).

When it comes to addressing entries via this client, it is possible to provide either the fully-qualified XTCE name in the format `/YSS/SIMULATOR/BatteryVoltage1` or an alias in the format `NAMESPACE/NAME`.

2.1 Reference

2.1.1 Client

Note: `MDBClient` instances are usually created via `YamcsClient.get_mdb()`:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
mdb = client.get_mdb(instance='simulator')
# ...
```

class `yamcs.mdb.client.MDBClient` (*ctx*, *instance*)

get_algorithm (*name*)

Gets a single algorithm by its unique name.

Parameters *name* (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.

Return type *Algorithm*

get_command (*name*)

Gets a single command by its unique name.

Parameters *name* (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.

Return type *Command*

get_container (*name*)

Gets a single container by its unique name.

Parameters *name* (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.

Return type *Container*

get_parameter (*name*)

Gets a single parameter by its name.

Parameters *name* (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.

Return type *Parameter*

get_space_system (*name*)

Gets a single space system by its unique name.

Parameters *name* (*str*) – A fully-qualified XTCE name

Return type *SpaceSystem*

list_algorithms (*page_size=None*)

Lists the algorithms visible to this client.

Algorithms are returned in lexicographical order.

Return type *Algorithm* iterator

list_commands (*page_size=None*)

Lists the commands visible to this client.

Commands are returned in lexicographical order.

Return type *Command* iterator

list_containers (*page_size=None*)

Lists the containers visible to this client.

Containers are returned in lexicographical order.

Return type *Container* iterator

list_parameters (*parameter_type=None, page_size=None*)

Lists the parameters visible to this client.

Parameters are returned in lexicographical order.

Parameters **parameter_type** (*str*) – The type of parameter

Return type *Parameter* iterator

list_space_systems (*page_size=None*)

Lists the space systems visible to this client.

Space systems are returned in lexicographical order.

Return type *SpaceSystem* iterator

2.1.2 Model

class `yamcs.mdb.model.Algorithm` (*proto*)

property **aliases**

List of (namespace, name) pairs, as 2-tuples

property **description**

Short description.

property **long_description**

Long description.

property **name**

Short name

property **qualified_name**

Full name (incl. space system)

class `yamcs.mdb.model.ArrayType` (*proto*)

property **arrayType**

Deprecated. Use `array_type`.

property **array_type**

In case the elements of an array of this type are also of type *array*, this returns type info of the elements' array type.

Note: This is an uncommon use case. Multi-dimensional arrays are more prevalent.

Type *ArrayType*

property dimensions

The number of dimensions in case of a multi-dimensional array.

property members

In case the elements of this array are of type *aggregate*, this returns an ordered list of its direct sub-members.

Type List[*Member*]

property name

Short name of this type.

class `yamcs.mdb.model.Command` (*proto*)

property abstract

Whether this is an abstract command. Abstract commands are intended for inheritance and cannot be issued directly.

property aliases

List of (namespace, name) pairs, as 2-tuples

property base_command

property description

Short description.

property long_description

Long description.

property name

Short name

property qualified_name

Full name (incl. space system)

property significance

class `yamcs.mdb.model.Container` (*proto*)

property aliases

List of (namespace, name) pairs, as 2-tuples

property description

Short description.

property long_description

Long description.

property name

Short name

property qualified_name

Full name (incl. space system)

class `yamcs.mdb.model.DataEncoding` (*proto*)

property bitlength

The size in bits

property encoding

Encoding detail

property little_endian

True if little-endian

property type

Raw type

class `yamcs.mdb.model.EnumValue` (*proto*)**property description**

State description

property label

String value

property value

Numeric value

class `yamcs.mdb.model.Member` (*proto*)

A member is a data structure for a specific field of a parent data type (either another member, or a parameter of type *aggregate*).

This is similar to C structs. The top-level of a member hierarchy is a parameter of type *aggregate*.

property arrayTypeDeprecated. Use `array_type`.**property array_type**In case this member is of type *array*, this returns array-specific type info.**Type** `ArrayType`**property members**In case this member is of type *aggregate*, this returns an ordered list of its direct sub-members.**Type** `List[Member]`**property name**

Short name

property type

Engineering type.

Type `str`**class** `yamcs.mdb.model.Parameter` (*proto*)

From XTCE:

A Parameter is a description of something that can have a value. It is not the value itself.

property aliases

List of (namespace, name) pairs, as 2-tuples

property arrayTypeDeprecated. Use `array_type`.**property array_type**In case this parameter is of type *array*, this returns array-specific type info.**Type** `ArrayType`

property data_encoding

Information on the raw encoding of this parameter, if applicable.

Type *DataEncoding*

property data_source

Specifies how this parameter originated (example: TELEMETERED)

Type *str*

property description

Short description.

property enum_values

In case this parameter is of type *enumeration*, this returns an ordered list of possible values.

Type List[*EnumValue*]

property long_description

Long description.

property members

In case this parameter is of type *aggregate*, this returns an ordered list of its direct members.

Type List[*Member*]

property name

Short name

property qualified_name

Full name (incl. space system)

property type

Engineering type.

Type *str*

property units

Engineering unit(s)

Type List[*str*]

class `yamcs.mdb.model.Significance` (*proto*)

property consequence_level

One of NONE, WATCH, WARNING, DISTRESS, CRITICAL or SEVERE.

property reason

Message attached to this significance.

class `yamcs.mdb.model.SpaceSystem` (*proto*)

From XTCE:

A SpaceSystem is a collection of SpaceSystem(s) including space assets, ground assets, multi-satellite systems and sub-systems. A SpaceSystem is the root element for the set of data necessary to monitor and command an arbitrary space device - this includes the binary decomposition of the data streams going into and out of a device.

property aliases

List of (namespace, name) pairs, as 2-tuples

property description

Short description.

property long_description

Long description.

property name

Short name

property qualified_name

Full name (incl. space system)

2.2 Snippets

Create an *MDBClient* for a specific instance:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
mdb = client.get_mdb(instance='simulator')
```

Print all space systems:

```
for space_system in mdb.list_space_systems():
    print(space_system)
```

Print all parameters of type float:

```
for parameter in mdb.list_parameters(parameter_type="float"):
    print(parameter)
```

Print all commands:

```
for command in mdb.list_commands():
    print(command)
```

Find a parameter by qualified name or alias:

```
p1 = mdb.get_parameter("/YSS/SIMULATOR/BatteryVoltage2")
print("Via qualified name:", p1)

p2 = mdb.get_parameter("MDB:OPS Name/SIMULATOR_BatteryVoltage2")
print("Via domain-specific alias:", p2)
```


TM/TC PROCESSING

The TM/TC API provides methods that you can use to programmatically interact with a TM/TC processor.

3.1 Reference

3.1.1 Client

Note: ProcessorClient instances are usually created via `YamcsClient.get_processor()`:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
processor = client.get_processor(instance='simulator',
                                processor='realtime')
# ...
```

class `yamcs.tmtc.client.ProcessorClient` (*ctx, instance, processor*)

Client object that groups operations linked to a specific processor.

acknowledge_alarm (*alarm, comment=None*)

Acknowledges a specific alarm.

Parameters

- **alarm** (*Alarm*) – Alarm instance
- **comment** (*str*) – Optional comment to associate with the state change.

clear_alarm (*alarm, comment=None*)

Clear an alarm.

Note: If the reason that caused the alarm is still present, a new alarm instance will be generated.

Parameters

- **alarm** (*Alarm*) – Alarm instance
- **comment** (*str*) – Optional comment to associate with the state change.

clear_alarm_ranges (*parameter*)

Removes all alarm limits for the specified parameter.

clear_calibrators (*parameter*)

Removes all calibrators for the specified parameter.

create_alarm_subscription (*on_data=None, timeout=60*)

Create a new alarm subscription.

Parameters

- **on_data** – (Optional) Function that gets called with *AlarmUpdate* updates.
- **timeout** (*float*) – The amount of seconds to wait for the request to complete.

Returns A Future that can be used to manage the background websocket subscription.

Return type *AlarmSubscription*

create_command_connection (*on_data=None, timeout=60*)

Creates a connection for issuing multiple commands and following up on their acknowledgment progress.

Note: This is a convenience method that merges the functionalities of *create_command_history_subscription()* with those of *issue_command()*.

Parameters

- **on_data** – Function that gets called with *CommandHistory* updates. Only commands issued from this connection are reported.
- **timeout** (*float*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription

Return type *CommandConnection*

create_command_history_subscription (*on_data=None, timeout=60*)

Create a new command history subscription.

Parameters

- **on_data** – (Optional) Function that gets called with *CommandHistory* updates.
- **timeout** (*float*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription

Return type *CommandHistorySubscription*

create_container_subscription (*containers, on_data=None, timeout=60*)

Create a new container subscription.

New in version 1.7.0: Compatible with Yamcs 5.5.0 onwards

Parameters

- **containers** (*Union[str, str[]]*) – Container names.
- **on_data** (*Optional[Callable[ContainerData]]*) – Function that gets called with *ContainerData* updates.
- **timeout** (*Optional[float]*) – The amount of seconds to wait for the request to complete.

Returns A Future that can be used to manage the background websocket subscription.

Return type *ContainerSubscription*

create_packet_subscription (*on_data*, *stream*='tm_realtime', *timeout*=60)

Create a new packet subscription.

New in version 1.6.6.

Parameters

- **on_data** (*Optional*[*Callable*[*Packet*]]) – Function that gets called with *Packet* updates.
- **stream** (*str*) – Stream to subscribe to.
- **timeout** (*Optional*[*float*]) – The amount of seconds to wait for the request to complete.

Returns A Future that can be used to manage the background websocket subscription.

Return type *WebSocketSubscriptionFuture*

create_parameter_subscription (*parameters*, *on_data*=None, *abort_on_invalid*=True, *update_on_expiration*=False, *send_from_cache*=True, *timeout*=60)

Create a new parameter subscription.

Parameters

- **parameters** (*str*[]) – Parameter names (or aliases).
- **on_data** – (Optional) Function that gets called with *ParameterData* updates.
- **abort_on_invalid** (*bool*) – If True an error is generated when invalid parameters are specified.
- **update_on_expiration** (*bool*) – If True an update is received when a parameter value has become expired. This update holds the same value as the last known valid value, but with status set to EXPIRED.
- **send_from_cache** (*bool*) – If True the last processed parameter value is sent from parameter cache. When False only newly processed parameters are received.
- **timeout** (*float*) – The amount of seconds to wait for the request to complete.

Returns A Future that can be used to manage the background websocket subscription.

Return type *ParameterSubscription*

get_parameter_value (*parameter*, *from_cache*=True, *timeout*=10)

Retrieve the current value of the specified parameter.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format NAMESPACE/NAME.
- **from_cache** (*bool*) – If False this call will block until a fresh value is received on the processor. If True the server returns the latest value instead (which may be None).
- **timeout** (*float*) – The amount of seconds to wait for a fresh value. (ignored if *from_cache*=True).

Return type *ParameterValue*

get_parameter_values (*parameters*, *from_cache*=True, *timeout*=10)

Retrieve the current value of the specified parameter.

Parameters

- **parameters** (*str*[]) – List of parameter names. These may be fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **from_cache** (*bool*) – If `False` this call will block until fresh values are received on the processor. If `True` the server returns the latest value instead (which may be `None`).
- **timeout** (*float*) – The amount of seconds to wait for a fresh values (ignored if `from_cache=True`).

Returns A list that matches the length and order of the requested list of parameters. Each entry contains either the returned parameter value, or `None`.

Return type *ParameterValue*[]

issue_command (*command*, *args=None*, *dry_run=False*, *comment=None*, *verification=None*, *extra=None*, *beta_args_v2=False*)

Issue the given command

Parameters

- **command** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **args** (*dict*) – named arguments (if the command requires these)
- **dry_run** (*bool*) – If `True` the command is not actually issued. This can be used to test if the server would generate errors when preparing the command (for example because an argument is missing).
- **comment** (*str*) – Comment attached to the command.
- **verification** (*VerificationConfig*) – Overrides to the default verification handling of this command.
- **extra** (*dict*) – Extra command options for interpretation by non-core extensions (custom preprocessor, datalinks, command listeners). Note that Yamcs will refuse command options that it does not know about. Extensions should therefore register available options.

Returns An object providing access to properties of the newly issued command.

Return type *IssuedCommand*

list_alarms (*start=None*, *stop=None*)

Lists the active alarms.

Remark that this does not query the archive. Only active alarms on the current processor are returned.

Parameters

- **start** (*datetime*) – Minimum trigger time of the returned alarms (inclusive)
- **stop** (*datetime*) – Maximum trigger time of the returned alarms (exclusive)

Return type *Iterable*[*Alarm*]

reset_alarm_ranges (*parameter*)

Reset all alarm limits for the specified parameter to their original MDB value.

reset_algorithm (*parameter*)

Reset the algorithm text to its original definition from MDB

Parameters **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.

reset_calibrators (*parameter*)

Reset all calibrators for the specified parameter to their original MDB value.

set_alarm_range_sets (*parameter, sets*)

Apply an ordered list of alarm range sets for the specified parameter. This replaces existing alarm sets (if any).

Each RangeSet may have a context, which indicates when its effects may be applied. Only the first matching set is applied.

A RangeSet with context `None` represents the *default* set of alarm ranges. There can be only one such set, and it is always applied at the end when no other set of contextual ranges is applicable.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **sets** (`RangeSet []`) – List of range sets (either contextual or not)

set_algorithm (*parameter, text*)

Change an algorithm text. Can only be performed on JavaScript or Python algorithms.

Parameters

- **text** (*string*) – new algorithm text (as it would appear in excel or XTCE)
- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.

set_calibrators (*parameter, calibrators*)

Apply an ordered set of calibrators for the specified parameter. This replaces existing calibrators (if any).

Each calibrator may have a context, which indicates when its effects may be applied. Only the first matching calibrator is applied.

A calibrator with context `None` is the *default* calibrator. There can be only one such calibrator, and is always applied at the end when no other contextual calibrator was applicable.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **calibrators** (`Calibrator []`) – List of calibrators (either contextual or not)

set_default_alarm_ranges (*parameter, watch=None, warning=None, distress=None, critical=None, severe=None, min_violations=1*)

Generate out-of-limit alarms for a parameter using the specified alarm ranges.

This replaces any previous default alarms on this parameter.

Note: Contextual range sets take precedence over the default alarm ranges. See [set_alarm_range_sets\(\)](#) for setting contextual range sets.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **watch** (*(float, float)*) – Range expressed as a tuple (`lo`, `hi`) where `lo` and `hi` are assumed exclusive.

- **warning** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **distress** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **critical** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **severe** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **min_violations** (*int*) – Minimum violations before an alarm is generated.

set_default_calibrator (*parameter, type, data*)

Apply a calibrator while processing raw values of the specified parameter. If there is already a default calibrator associated to this parameter, that calibrator gets replaced.

Note: Contextual calibrators take precedence over the default calibrator See *set_calibrators()* for setting contextual calibrators.

Two types of calibrators can be applied:

- Polynomial calibrators apply a polynomial expression of the form: $y = a + bx + cx^2 + \dots$.
The *data* argument must be an array of floats [a, b, c, ...].
- Spline calibrators interpolate the raw value between a set of points which represent a linear curve.
The *data* argument must be an array of [x, y] points.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format NAMESPACE/NAME.
- **type** (*str*) – One of *polynomial* or *spline*.
- **data** – Calibration definition for the selected type.

set_parameter_value (*parameter, value*)

Sets the value of the specified parameter.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format NAMESPACE/NAME.
- **value** – The value to set

set_parameter_values (*values*)

Sets the value of multiple parameters.

Parameters values (*dict*) – Values keyed by parameter name. This name can be either a fully-qualified XTCE name or an alias in the format NAMESPACE/NAME.

shelve_alarm (*alarm, comment=None*)

Shelve an alarm.

Parameters

- **alarm** (*Alarm*) – Alarm instance

- **comment** (*str*) – Optional comment to associate with the state change.

unshelve_alarm (*alarm*, *comment=None*)

Unshelve an alarm.

Parameters

- **alarm** (*Alarm*) – Alarm instance
- **comment** (*str*) – Optional comment to associate with the state change.

class `yamcs.tmtc.client.CommandConnection` (*manager*, *tmtc_client*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object providing access to the acknowledgment progress of command updates.

Only commands issued from this object are monitored.

Initializes the future. Should not be called by clients.

issue (*command*, *args=None*, *dry_run=False*, *comment=None*, *verification=None*, *extra=None*, *beta_args_v2=False*)

Issue the given command

Parameters

- **command** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **args** (*dict*) – Named arguments (if the command requires these)
- **dry_run** (*bool*) – If `True` the command is not actually issued. This can be used to test if the server would generate errors when preparing the command (for example because an argument is missing).
- **comment** (*str*) – Comment attached to the command.
- **verification** (*VerificationConfig*) – Overrides to the default verification handling of this command.
- **extra** (*dict*) – Extra command options for interpretation by non-core extensions (custom preprocessor, datalinks, command listeners). Note that Yamcs will refuse command options that it does not know about. Extensions should therefore register available options.

Returns An object providing access to properties of the newly issued command and updated according to command history updates.

Return type *MonitoredCommand*

class `yamcs.tmtc.client.AlarmSubscription` (*manager*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object representing an alarm subscription.

A subscription object stores the currently active alarms.

Initializes the future. Should not be called by clients.

get_alarm (*name*)

Returns the alarm state associated with a specific named alarm from local cache.

Parameters **name** (*str*) – Fully-qualified name

Return type *Alarm*

list_alarms ()

Returns a snapshot of all active alarms.

Return type *Alarm*[]

class `yamcs.tmtc.client.CommandHistorySubscription` (*manager*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object providing access to command history updates.

This object buffers all received command history. This is needed to stitch together incremental command history events.

If you expect to receive a lot of command history updates you should periodically clear local cache via `clear_cache()`. In future work, we may add automated buffer management within configurable watermarks.

Warning: If command history updates are received for commands that are not currently in the local cache, the returned information may be incomplete.

Initializes the future. Should not be called by clients.

clear_cache ()

Clears local command history cache.

get_command_history (*issued_command*)

Gets locally cached CommandHistory for the specified command.

Parameters **issued_command** (*IssuedCommand*) – object representing a previously issued command.

Return type *CommandHistory*

class `yamcs.tmtc.client.ParameterSubscription` (*manager*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object representing a subscription of zero or more parameters.

A subscription object stores the last received value of each subscribed parameter.

Initializes the future. Should not be called by clients.

add (*parameters*, *abort_on_invalid=True*, *send_from_cache=True*)

Add one or more parameters to this subscription.

Parameters

- **parameters** (*Union[str, str[]]*) – Parameter(s) to be added
- **abort_on_invalid** (*bool*) – If `True` one invalid parameter means any other parameter in the request will also not be added to the subscription.
- **send_from_cache** (*bool*) – If `True` the last processed parameter value is sent from parameter cache. When `False` only newly processed parameters are received.

delivery_count = `None`

The number of parameter deliveries.

get_value (*parameter*)

Returns the last value of a specific parameter from local cache.

Parameters **parameter** (*string*) – Parameter name.

Return type *ParameterValue*

remove (*parameters*)

Remove one or more parameters from this subscription.

Parameters *parameters* (*Union[str, str[]]*) – Parameter(s) to be removed

value_cache = **None**

Value cache keyed by parameter name.

3.1.2 Model

class `yamcs.tmtc.model.Acknowledgment` (*name, time, status, message*)

is_terminated ()

message = **None**

Message of this acknowledgment.

name = **None**

Name of this acknowledgment.

status = **None**

Status of this acknowledgment.

time = **None**

Last update time of this acknowledgment.

class `yamcs.tmtc.model.Alarm` (*proto*)

property `acknowledge_message`

Comment provided when acknowledging the alarm.

property `acknowledge_time`

Processor time when the alarm was acknowledged.

Type `datetime`

property `acknowledged_by`

Username of the acknowledger.

property `count`

Total number of samples while this alarm is active.

property `is_acknowledged`

True if this alarm has been acknowledged.

property `is_latched`

True if this alarm is currently latched.

property `is_latching`

True if this is a latching alarm. A latching alarm returns to normal only when the operator resets it

property `is_ok`

True if this alarm is currently 'inactive'.

For a non-latching alarm this is identical to `is_process_ok()`. A latching alarm is only OK if it has returned to normal and was acknowledged.

property `is_process_ok`

True if the process that caused this alarm is OK. For example: parameter back within limits.

For a non-latching alarm this is identical to `is_ok()`.

property is_shelved

True if this alarm has been shelved.

property name

Fully-qualified name of the source of this alarm.

property sequence_number

Sequence number for this specific alarm instance. This allows ensuring that operations (such as acknowledgment) are done on the expected alarm instance.

property severity

property trigger_time

Processor time when the alarm was triggered.

Type `datetime`

property violation_count

Number of violating samples while this alarm is active.

class `yamcs.tmtc.model.AlarmUpdate` (*proto*)

Object received through callbacks when subscribing to alarm updates.

property alarm

Latest alarm state.

Type `Alarm`

property update_type

Type of update.

class `yamcs.tmtc.model.Calibrator` (*context, type, data*)

A calibrator that may be applied to a numeric raw value.

Two types of calibrators can be applied:

- **Polynomial calibrators apply a polynomial expression of the form:** $y = a + bx + cx^2 + \dots$
The *data* argument must be an array of floats [*a*, *b*, *c*, ...].
- **Spline calibrators interpolate the raw value between a set of points** which represent a linear curve.
The *data* argument must be an array of [*x*, *y*] points.

Parameters

- **context** (*str*) – Condition under which this calibrator may be applied. The value `None` indicates the default calibrator which is only applied if no contextual calibrators match.
- **type** (*str*) – One of `polynomial` or `spline`.
- **data** – Calibration definition for the selected type.

`POLYNOMIAL = 'polynomial'`

`SPLINE = 'spline'`

class `yamcs.tmtc.model.CommandHistory` (*proto*)

property acknowledgments

All acknowledgments by name.

Returns Acknowledgments keyed by name.

Return type `OrderedDict`

property binary

Binary representation of the command.

property comment

Optional user comment attached when issuing the command.

property error

Error message in case the command failed.

generation_time = None

The generation time as set by Yamcs

Type `datetime`

is_complete()

Returns whether this command is complete. A command can be completed, yet still failed.

is_failure()

Returns True if the command failed.

is_success()

Returns True if the command has completed successfully.

property name

Name of the command.

property origin

The origin of this command. This is often empty, but may also be a hostname.

property sequence_number

The sequence number of this command. This is the sequence number assigned by the issuing client.

property source

String representation of the command.

property username

Username of the issuer.

class `yamcs.tmtc.model.ContainerData` (*proto*)

property binary

Raw binary

property generation_time

The time when this container's packet was generated (packet time).

Type `datetime`

property name

The name of the container.

property reception_time

The time when this container's packet was received by Yamcs.

Type `datetime`

class `yamcs.tmtc.model.EventAlarm` (*proto*)

An alarm triggered by an event.

property current_event

Latest event for this alarm

Type `Event`

property most_severe_event

First event that invoked the highest severity level of this alarm

Type *Event*

property trigger_event

Event that originally triggered the alarm

Type *Event*

class `yamcs.tmtc.model.IssuedCommand` (*proto*)

property binary

Binary representation of this command.

property generation_time

The generation time as set by Yamcs.

Type *datetime*

property hex

Hexadecimal string representation of this command.

property id

A unique identifier for this command.

property name

The fully-qualified name of this command.

property origin

The origin of this command. Usually the IP address of the issuer.

property queue

The name of the queue that this command was assigned to.

property sequence_number

The sequence number of this command. This is the sequence number assigned by the issuing client.

property source

String representation of this command.

property username

The username of the issuer.

class `yamcs.tmtc.model.MonitoredCommand` (*proto*)

Represent an instance of an issued command that is updated throughout the acknowledgment process.

Objects of this class are owned by a *CommandConnection* instance.

property acknowledgments

All acknowledgments by name.

Returns Acknowledgments keyed by name.

Return type *OrderedDict*

property attributes

await_acknowledgment (*name, timeout=None*)

Waits for the result of a specific acknowledgment.

Parameters

- **name** (*str*) – The name of the acknowledgment. Standard names are `Acknowledge_Queued`, `Acknowledge_Released` and `Acknowledge_Sent`. Others depend on specific link types.
- **timeout** (*float*) – The amount of seconds to wait.

Return type *Acknowledgment*

await_complete (*timeout=None*)

Wait for the command to be completed.

Parameters **timeout** (*float*) – The amount of seconds to wait.

property comment

Optional user comment attached when issuing the command.

property error

Error message in case the command failed.

is_complete ()

Returns whether this command is complete. A command can be completed, yet still failed.

is_success ()

Returns true if this command was completed successfully.

class `yamcs.tmtc.model.Packet` (*proto*)

property binary

Raw binary of this packet

property generation_time

The time when the packet was generated (packet time).

Type *datetime*

property name

The name of the packet. When using XTCE extraction this is the fully-qualified name of the first container in the hierarchy that this packet maps to.

property reception_time

The time when the packet was received by Yamcs.

Type *datetime*

property sequence_number

The sequence number of the packet. This is usually decoded from the packet.

class `yamcs.tmtc.model.ParameterAlarm` (*proto*)

An alarm triggered by a parameter that went out of limits.

property current_value

Latest parameter value for this alarm.

Type *ParameterValue*

property most_severe_value

First parameter value that invoked the highest severity level of this alarm.

Type *ParameterValue*

property trigger_value

Parameter value that originally triggered the alarm

Type *ParameterValue*

class `yamcs.tmtc.model.ParameterData` (*proto, mapping=None*)

get_value (*parameter*)

Returns the value of a specific parameter. Or None if the parameter is not included in this update.

Parameters `parameter` (*string*) – Parameter name.

Return type `ParameterValue`

property `parameters`

Type `List[ParameterValue]`

class `yamcs.tmtc.model.ParameterValue` (*proto, id=None*)

property `eng_value`

The engineering (calibrated) value.

property `generation_time`

The time when the parameter was generated. If the parameter was extracted from a packet, this usually returns the packet time.

Type `datetime`

property `monitoring_result`

property `name`

An identifying name for the parameter value. Typically this is the fully-qualified XTCE name, but it may also be an alias depending on how the parameter update was requested.

property `processing_status`

property `range_condition`

If the value is out of limits, this indicates LOW or HIGH.

property `raw_value`

The raw (uncalibrated) value.

property `reception_time`

The time when the parameter value was received by Yamcs.

Type `datetime`

property `validity_duration`

How long this parameter value is valid.

Type `timedelta`

property `validity_status`

class `yamcs.tmtc.model.RangeSet` (*context, watch=None, warning=None, distress=None, critical=None, severe=None, min_violations=1*)

A set of alarm range that apply in a specific context.

Parameters

- **context** (*str*) – Condition under which this range set is applicable. The value None indicates the default range set which is only applicable if no contextual sets match.
- **watch** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **warning** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.

- **distress** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **critical** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **severe** (*(float, float)*) – Range expressed as a tuple (lo, hi) where lo and hi are assumed exclusive.
- **min_violations** (*int*) – Minimum violations before an alarm is generated.

class `yamcs.tmtc.model.VerificationConfig`

Contains overrides to the default verification handling of Yamcs.

disable (*verifier=None*)

Disable verification.

Parameters **verifier** (*str*) – Name of a specific verifier to disable. If unspecified all verifiers are disabled.

modify_check_window (*verifier, start=None, stop=None*)

Set or override the check window.

Depending on the Mission Database configuration, the time may be relative to either the command release or a preceding verifier.

Parameters

- **verifier** (*str*) – Name of the verifier
- **start** (*float*) – Window start time (relative, in seconds)
- **stop** (*float*) – Window stop time (relative, in seconds)

3.2 Snippets

Create a *ProcessorClient* for a specific processor:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
processor = client.get_processor(instance='simulator', processor='realtime')
```

3.2.1 Read/Write Parameters

Read a single value. This returns the latest value from server cache.

```
pval = processor.get_parameter_value("/YSS/SIMULATOR/BatteryVoltage1")
print(pval)
```

Read a single value, but block until a fresh value could be processed:

```
pval = processor.get_parameter_value(
    "/YSS/SIMULATOR/BatteryVoltage2", from_cache=False, timeout=5
)
print(pval)
```

Read the latest value of multiple parameters at the same time:

```
pvals = processor.get_parameter_values(
    ["/YSS/SIMULATOR/BatteryVoltage1", "/YSS/SIMULATOR/BatteryVoltage2"]
)
print("battery1", pvals[0])
print("battery2", pvals[1])
```

Set the value of a parameter. Only some types of parameters can be written to. This includes software parameters (local to Yamcs) and parameters that are linked to an external system (such as a simulator).

```
processor.set_parameter_value("/YSS/SIMULATOR/AllowCriticalTC1", True)
```

Set the value of multiple parameters:

```
processor.set_parameter_values(
    {
        "/YSS/SIMULATOR/AllowCriticalTC1": False,
        "/YSS/SIMULATOR/AllowCriticalTC2": False,
    }
)
```

3.2.2 Parameter Subscription

Poll latest values from a subscription:

```
subscription = processor.create_parameter_subscription(
    ["/YSS/SIMULATOR/BatteryVoltage1"]
)

sleep(5)
print("Latest value:")
print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage1"))

sleep(5)
print("Latest value:")
print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage1"))
```

Receive *ParameterData* callbacks whenever one or more of the subscribed parameters have been updated:

```
def print_data(data):
    for parameter in data.parameters:
        print(parameter)

processor.create_parameter_subscription(
    "/YSS/SIMULATOR/BatteryVoltage1", on_data=print_data
)
sleep(5) # Subscription is non-blocking
```

Create and modify a parameter subscription:

```
subscription = processor.create_parameter_subscription(
    ["/YSS/SIMULATOR/BatteryVoltage1"]
)

sleep(5)
```

(continues on next page)

(continued from previous page)

```

print("Adding extra items to the existing subscription...")
subscription.add(
    [
        "/YSS/SIMULATOR/Alpha",
        "/YSS/SIMULATOR/BatteryVoltage2",
        "MDB:OPS Name/SIMULATOR_PrimBusVoltage1",
    ]
)

sleep(5)

print("Shrinking subscription...")
subscription.remove("/YSS/SIMULATOR/Alpha")

print("Cancelling the subscription...")
subscription.cancel()

print("Last values from cache:")
print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage1"))
print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage2"))
print(subscription.get_value("/YSS/SIMULATOR/Alpha"))
print(subscription.get_value("MDB:OPS Name/SIMULATOR_PrimBusVoltage1"))

```

3.2.3 Commanding

Issue a command (fire-and-forget):

```

command = processor.issue_command(
    "/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF", args={"voltage_num": 1}
)
print("Issued", command)

```

To monitor acknowledgments, establish a command connection first. Commands issued from this connection are automatically updated with progress status:

```

conn = processor.create_command_connection()

command = conn.issue("/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF", args={"voltage_num": 1})

ack = command.await_acknowledgment("Acknowledge_Sent")
print(ack.status)

```

The default Yamcs-local acknowledgments are:

- Acknowledge_Queued
- Acknowledge_Released
- Acknowledge_Sent

Custom telemetry verifiers or command links may cause additional acknowledgments to be generated.

If configured, command completion can also be monitored:

```
conn = processor.create_command_connection()

command1 = conn.issue("/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF", args={"voltage_num": 1})

# Issue 2nd command only if the previous command was completed successfully.
command1.await_complete()
if command1.is_success():
    conn.issue("/YSS/SIMULATOR/SWITCH_VOLTAGE_ON", args={"voltage_num": 1})
else:
    print("Command 1 failed:", command1.error)
```

3.2.4 Alarm Monitoring

Receive *AlarmUpdate* callbacks:

```
def callback(alarm_update):
    print("Alarm Update:", alarm_update)

processor.create_alarm_subscription(callback)
```

Acknowledge all active alarms:

```
for alarm in processor.list_alarms():
    if not alarm.is_acknowledged:
        processor.acknowledge_alarm(alarm, comment="false alarm")
```

The Archive API provides methods that you can use to programmatically retrieve the content of a Yamcs Archive.

4.1 Reference

4.1.1 Client

Note: ArchiveClient instances are usually created via `YamcsClient.get_archive()`:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
archive = client.get_archive(instance='simulator')
# ...
```

class `yamcs.archive.client.ArchiveClient` (*ctx, instance*)

create_stream_subscription (*stream, on_data, timeout=60*)

Create a new stream subscription.

Parameters

- **stream** (*str*) – The name of the stream.
- **on_data** – Function that gets called with *StreamData* updates.
- **timeout** (*float*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription

Return type *WebSocketSubscriptionFuture*

dump_table (*table, chunk_size=32768*)

execute_sql (*statement*)

Executes a single SQL statement.

Parameters **statement** – SQL string

Returns A result set for consuming rows

Return type *ResultSet*

export_packets (*name=None, start=None, stop=None, chunk_size=1024*)

Export raw packets.

Packets are sorted by generation time and sequence number.

Parameters

- **name** (*str*) – Archived name of the packet
- **start** (*datetime*) – Minimum generation time of the returned packets (inclusive)
- **stop** (*datetime*) – Maximum generation time of the returned packets (exclusive)

Return type An iterator over received chunks

get_packet (*generation_time, sequence_number*)

Gets a single packet by its identifying key (gentime, seqNum).

Parameters

- **generation_time** (*datetime*) – When the packet was generated (packet time)
- **sequence_number** (*int*) – Sequence number of the packet

Return type *Packet*

get_stream (*stream*)

Gets a single stream.

Parameters **stream** (*str*) – The name of the stream.

Return type *Stream*

get_table (*table*)

Gets a single table.

Parameters **table** (*str*) – The name of the table.

Return type *Table*

list_command_histogram (*name=None, start=None, stop=None, merge_time=2*)

Reads command-related index records between the specified start and stop time.

Each iteration returns a chunk of chronologically-sorted records.

Parameters **merge_time** (*float*) – Maximum gap in seconds before two consecutive index records are merged together.

Return type *Iterable[IndexGroup]*

list_command_history (*command=None, start=None, stop=None, page_size=500, descending=False*)

Reads command history entries between the specified start and stop time.

Parameters

- **command** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **start** (*datetime*) – Minimum generation time of the returned command history entries (inclusive)
- **stop** (*datetime*) – Maximum generation time of the returned command history entries (exclusive)
- **page_size** (*int*) – Page size of underlying requests. Higher values imply less overhead, but risk hitting the maximum message size limit.

- **descending** (*bool*) – If set to `True` results are fetched in reverse order (most recent first).

Return type `Iterable[CommandHistory]`

list_completeness_index (*start=None, stop=None*)

Reads completeness index records between the specified start and stop time.

Each iteration returns a chunk of chronologically-sorted records.

Return type `Iterable[IndexGroup]`

list_event_histogram (*source=None, start=None, stop=None, merge_time=2*)

Reads event-related index records between the specified start and stop time.

Each iteration returns a chunk of chronologically-sorted records.

Parameters **merge_time** (*float*) – Maximum gap in seconds before two consecutive index records are merged together.

Return type `Iterable[IndexGroup]`

list_event_sources ()

Returns the existing event sources.

Return type `Iterable[str]`

list_events (*source=None, severity=None, text_filter=None, start=None, stop=None, page_size=500, descending=False*)

Reads events between the specified start and stop time.

Events are sorted by generation time, source, then sequence number.

Parameters

- **source** (*str*) – The source of the returned events.
- **severity** (*str*) – The minimum severity level of the returned events. One of `INFO`, `WATCH`, `WARNING`, `DISTRESS`, `CRITICAL` or `SEVERE`.
- **text_filter** (*str*) – Filter the text message of the returned events
- **start** (*datetime*) – Minimum start date of the returned events (inclusive)
- **stop** (*datetime*) – Maximum start date of the returned events (exclusive)
- **page_size** (*int*) – Page size of underlying requests. Higher values imply less overhead, but risk hitting the maximum message size limit.
- **descending** (*bool*) – If set to `True` events are fetched in reverse order (most recent first).

Return type `Iterable[Event]`

list_packet_histogram (*name=None, start=None, stop=None, merge_time=2*)

Reads packet-related index records between the specified start and stop time.

Each iteration returns a chunk of chronologically-sorted records.

Parameters **merge_time** (*float*) – Maximum gap in seconds before two consecutive index records are merged together.

Return type `Iterable[IndexGroup]`

list_packet_names ()

Returns the existing packet names.

Return type `Iterable[str]`

list_packets (*name=None, start=None, stop=None, page_size=500, descending=False*)

Reads packet information between the specified start and stop time.

Packets are sorted by generation time and sequence number.

Parameters

- **name** (*str*) – Archived name of the packet
- **start** (*datetime*) – Minimum generation time of the returned packets (inclusive)
- **stop** (*datetime*) – Maximum generation time of the returned packets (exclusive)
- **page_size** (*int*) – Page size of underlying requests. Higher values imply less overhead, but risk hitting the maximum message size limit.
- **descending** (*bool*) – If set to `True` packets are fetched in reverse order (most recent first).

Return type `Iterable[Packet]`

list_parameter_ranges (*parameter, start=None, stop=None, min_gap=None, max_gap=None, min_range=None, max_values=100, parameter_cache='realtime'*)

Returns parameter ranges between the specified start and stop time.

Each range indicates an interval during which this parameter's value was uninterrupted and unchanged.

Ranges are a good fit for retrieving the value of a parameter that does not change frequently. For example an on/off indicator or some operational status. Querying ranges will then induce much less overhead than manually processing the output of `list_parameter_values()` would.

The maximum number of returned ranges is limited to 500.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **start** (*Optional[datetime]*) – Minimum generation time of the considered values (inclusive)
- **stop** (*Optional[datetime]*) – Maximum generation time of the considered values (exclusive)
- **min_gap** – Time in seconds. Any gap (detected based on parameter expiration) smaller than this will be ignored. However if the parameter changes value, the ranges will still be split.
- **max_gap** (*Optional[float]*) – Time in seconds. If the distance between two subsequent parameter values is bigger than this value (but smaller than the parameter expiration), then an artificial gap is created. This also applies if there is no expiration defined for the parameter.
- **min_range** (*Optional[float]*) – Time in seconds. Minimum duration of returned ranges. If multiple values occur within the range, the most frequent can be accessed using the `entries` property.
- **max_values** – Maximum number of unique values, tallied across the full requested range. Use this in combination with `min_range` to further optimize for transfer size. This value is limited to 100 at most.
- **parameter_cache** (*Optional[str]*) – Specify the name of the processor who's parameter cache is merged with already archived values. To disable results from the parameter cache, set this to `None`.

Return type *ParameterRange*[]

list_parameter_values (*parameter*, *start=None*, *stop=None*, *page_size=500*, *descending=False*, *parameter_cache='realtime'*, *source='ParameterArchive'*)

Reads parameter values between the specified start and stop time.

Note: This method will send out multiple requests when more than *page_size* values are queried. For large queries, consider using *stream_parameter_values()* instead, it uses server-streaming based on a single request, and supports downloading the values of multiple parameter at the same time.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **start** (*datetime*) – Minimum generation time of the returned values (inclusive)
- **stop** (*datetime*) – Maximum generation time of the returned values (exclusive)
- **page_size** (*int*) – Page size of underlying requests. Higher values imply less overhead, but risk hitting the maximum message size limit.
- **descending** (*bool*) – If set to `True` values are fetched in reverse order (most recent first).
- **parameter_cache** (*str*) – Specify the name of the processor who's parameter cache is merged with already archived values. To disable results from the parameter cache, set this to `None`.
- **source** (*str*) – Specify how to retrieve parameter values. By default this uses the `ParameterArchive` which is optimized for retrieval. For Yamcs instances that do not enable the `ParameterArchive`, you can still get results by specifying `replay` as the source. Replay requests take longer to return because the data needs to be reprocessed.

Return type *Iterable*[*ParameterValue*]

list_processed_parameter_group_histogram (*group=None*, *start=None*, *stop=None*, *merge_time=20*)

Reads index records related to processed parameter groups between the specified start and stop time.

Each iteration returns a chunk of chronologically-sorted records.

Parameters *merge_time* (*float*) – Maximum gap in seconds before two consecutive index records are merged together.

Return type *Iterable*[*IndexGroup*]

list_processed_parameter_groups ()

Returns the existing parameter groups.

Return type *Iterable*[*str*]

list_streams ()

Returns the existing streams.

Streams are returned in lexicographical order.

Return type *Iterable*[*Stream*]

list_tables ()

Returns the existing tables.

Tables are returned in lexicographical order.

Return type `Iterable[Table]`

load_table (*table*, *data*, *chunk_size=32768*)

rebuild_histogram (*table*, *start=None*, *stop=None*)

Rebuilds the histogram for a table. This may be necessary for example after bulk loading data.

The rebuild may be constrained by using the `start` and `stop` parameters. When specified all partitions overlapping this range are reconsidered.

Note: Histogram rebuilds run synchronously: this method will await the outcome.

Parameters

- **table** (*str*) – The name of the table
- **start** (*Optional[datetime]*) – Start time
- **stop** (*Optional[datetime]*) – Stop time

rebuild_parameter_archive (*start*, *stop*)

Rebuilds the Parameter Archive.

The rebuild must be constrained by using the `start` and `stop` parameters. This values are only hints to the Parameter Archive, which will extend the requested range based on archive segmentation.

Note: Rebuilds run as an asynchronous operation: this method will not await the outcome.

Parameters

- **start** (*datetime*) – Start time
- **stop** (*datetime*) – Stop time

sample_parameter_values (*parameter*, *start=None*, *stop=None*, *sample_count=500*, *parameter_cache='realtime'*, *source='ParameterArchive'*)

Returns parameter samples.

The query range is split in sample intervals of equal length. For each interval a *Sample* is returned which describes the min, max, count and avg during that interval.

Note that sample times are determined without considering the actual parameter values. Two separate queries with equal start/stop arguments will always return the same number of samples with the same timestamps. This is done to ease merging of multiple sample series. You should always be explicit about the `start` and `stop` times when relying on this property.

Parameters

- **parameter** (*str*) – Either a fully-qualified XTCE name or an alias in the format `NAMESPACE/NAME`.
- **start** (*datetime*) – Minimum generation time of the sampled parameter values (inclusive). If not set this defaults to one hour ago.
- **stop** (*datetime*) – Maximum generation time of the sampled parameter values (exclusive). If not set this defaults to the current time.
- **sample_count** (*int*) – The number of returned samples.

- **parameter_cache** (*str*) – Specify the name of the processor who’s parameter cache is merged with already archived values. To disable results from the parameter cache, set this to `None`.
- **source** (*str*) – Specify how to retrieve parameter values. By default this uses the `ParameterArchive` which is optimized for retrieval. For Yamcs instances that do not enable the `ParameterArchive`, you can still get results by specifying `replay` as the source. Replay requests take longer to return because the data needs to be reprocessed.

Return type *Sample*[]

stream_parameter_values (*parameters*, *start=None*, *stop=None*, *chunk_size=32768*)

Reads parameter values between the specified start and stop time.

Value updates are emitted for each unique generation time within the queried range. If one of the parameters does not have a value for a specific generation time, it is not included in the update.

Parameters

- **parameters** (*Union[str, str[]]*) – Parameter(s) to be queried.
- **start** (*datetime*) – Minimum generation time of the returned values (inclusive)
- **stop** (*datetime*) – Maximum generation time of the returned values (exclusive)

Returns

rtype `~collections.abc.Iterable[ParameterData]`

4.1.2 Model

class `yamcs.archive.model.ColumnData` (*proto*)

property name

Column name.

property value

Value for this column.

class `yamcs.archive.model.IndexGroup` (*proto*)

Group of index records that represent the same type of underlying objects.

property name

Name associated with this group. The meaning is defined by the objects represented by this index. For example:

- In an index of events, index records are grouped by `source`.
- In an index of packets, index records are grouped by `packet name`.

property records

Index records within this group

Type `List[IndexRecord]`

class `yamcs.archive.model.IndexRecord` (*proto*)

Represents a block of uninterrupted data (derived from the index definition for the type of underlying objects, in combination with the requested `merge_time`).

property count

Number of underlying objects this index record represents

property start

Start time of the record

Type `datetime`

property stop

Stop time of the record

Type `datetime`

class `yamcs.archive.model.ParameterRange` (*proto*)

Indicates an interval during which a parameter's value was uninterrupted and unchanged.

property eng_value

The engineering (calibrated) value within this range.

If the request was made using `min_range` option, this will be the most-frequent value only. Retrieve the complete distribution using the `entries` attribute.

property entries

Value distribution within this range.

Unless the request was made using `min_range` option, there should be only one entry only.

Type `List[ParameterRangeEntry]`

property parameter_count

The total number of parameter values within this range.

property start

Start time of this range (inclusive).

Type `datetime`

property stop

Stop time of this range (exclusive).

Type `datetime`

class `yamcs.archive.model.ParameterRangeEntry` (*proto*)

Value holder for an engineering value and its number of appearances within a `ParameterRange`.

property eng_value

The engineering (calibrated) value.

property parameter_count

The number of received parameter values during this range.

class `yamcs.archive.model.ResultSet` (*response*)

Provides capability to consume the rows returned by a SQL query, or access related information.

property column_types

Column types.

Type `str[]`

property columns

Column names. This returns `None` as long as no row has been consumed yet.

Type `str[]`

class `yamcs.archive.model.Sample` (*proto*)

Provides aggregation properties over a range of a parameter's values.

property avg

Average value.

property max
Maximum value.

property min
Minimum value.

property parameter_count
The number of parameter values this sample represents.

property time
Sample time.

Type `datetime`

class `yamcs.archive.model.Stream(proto)`

property name
Stream name.

class `yamcs.archive.model.StreamData(proto)`

property columns
Tuple columns.

property stream
Stream name.

class `yamcs.archive.model.Table(proto)`

property name
Table name.

4.2 Snippets

Create an *ArchiveClient* for a specific instance:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
archive = client.get_archive(instance='simulator')
```

4.2.1 Packet Retrieval

Print the last 10 packets:

```
for packet in islice(archive.list_packets(descending=True), 0, 10):
    print(packet)
```

Print available range of archived packets:

```
first_packet = next(iter(archive.list_packets()))
last_packet = next(iter(archive.list_packets(descending=True)))
print("First packet:", first_packet)
print("Last packet:", last_packet)
```

(continues on next page)

(continued from previous page)

```
td = last_packet.generation_time - first_packet.generation_time
print("Timespan:", td)
```

Iterate a specific range of packets:

```
now = datetime.now(tz=timezone.utc)
start = now - timedelta(hours=1)

total = 0
for packet in archive.list_packets(start=start, stop=now):
    total += 1
    # print(packet)
print("Found", total, "packets in range")
```

Download raw packet binary to a file:

```
now = datetime.now(tz=timezone.utc)
start = now - timedelta(hours=1)

with open("/tmp/dump.raw", "wb") as f:
    for chunk in archive.export_packets(start=start, stop=now):
        f.write(chunk)
```

4.2.2 Parameter Retrieval

Retrieve the last 10 values of a parameter:

```
iterable = archive.list_parameter_values(
    "/YSS/SIMULATOR/BatteryVoltage1", descending=True
)
for pval in islice(iterable, 0, 10):
    print(pval)
```

Iterate a specific range of values:

```
now = datetime.now(tz=timezone.utc)
start = now - timedelta(hours=1)

total = 0
for pval in archive.list_parameter_values(
    "/YSS/SIMULATOR/BatteryVoltage1", start=start, stop=now
):
    total += 1
    # print(pval)
print("Found", total, "parameter values in range")
```

Iterate values of multiple parameters by unique generation time:

```
now = datetime.now(tz=timezone.utc)
start = now - timedelta(hours=1)

total = 0
for pdata in archive.stream_parameter_values(
    ["/YSS/SIMULATOR/BatteryVoltage1", "/YSS/SIMULATOR/BatteryVoltage2"],
```

(continues on next page)

(continued from previous page)

```

    start=start,
    stop=now,
):
    total += 1
    # print(pdata)
print("Found", total, "updates in range")

```

4.2.3 Event Retrieval

Iterate a specific range of events:

```

now = datetime.now(tz=timezone.utc)
start = now - timedelta(hours=1)

total = 0
for event in archive.list_events(start=start, stop=now):
    total += 1
    # print(event)
print("Found", total, "events in range")

```

4.2.4 Command Retrieval

Retrieve the last 10 issued commands:

```

iterable = archive.list_command_history(descending=True)
for entry in islice(iterable, 0, 10):
    print(entry)

```

4.2.5 Histogram Retrieval

Print the number of packets grouped by packet name:

```

for name in archive.list_packet_names():
    packet_count = 0
    for group in archive.list_packet_histogram(name):
        for rec in group.records:
            packet_count += rec.count
    print(f" {name: <40} {packet_count: >20}")

```

Print the number of events grouped by source:

```

for source in archive.list_event_sources():
    event_count = 0
    for group in archive.list_event_histogram(source):
        for rec in group.records:
            event_count += rec.count
    print(f" {source: <40} {event_count: >20}")

```

Print the number of processed parameter frames grouped by group name:

```
for group in archive.list_processed_parameter_groups():
    frame_count = 0
    for pp_group in archive.list_processed_parameter_group_histogram(group):
        for rec in pp_group.records:
            frame_count += rec.count
    print(f" {group: <40} {frame_count: >20}")
```

Print the number of commands grouped by name:

```
mdb = client.get_mdb(instance="simulator")
for command in mdb.list_commands():
    total = 0
    for group in archive.list_command_histogram(command.qualified_name):
        for rec in group.records:
            total += rec.count
    print(f" {command.qualified_name: <40} {total: >20}")
```


LINK MANAGEMENT

The Link API provides methods that you can use to programmatically interact with a Yamcs link.

5.1 Reference

5.1.1 Client

Note: `LinkClient` instances are usually created via `YamcsClient.get_link()`:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
link = client.get_link(instance='simulator', link='udp-in')
# ...
```

class `yamcs.link.client.LinkClient` (*ctx, instance, link*)

Client object that groups operations for a specific link.

create_cop1_subscription (*on_data, timeout=60*)

Create a new subscription for receiving status of the COPI link.

This method returns a future, then returns immediately. Stop the subscription by canceling the future.

Parameters

- **on_data** (*Optional[Callable[Cop1Status]]*) – Function that gets called on each *Cop1Status*.
- **timeout** (*Optional[float]*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription.

Return type *Cop1Subscription*

disable_cop1 (*bypass_all=True*)

Disable COPI for a data link.

Parameters **bypass_all** (*bool*) – All frames have bypass activated (i.e. they will be BD frames)

disable_link ()

Disables this link.

enable_link()

Enables this link.

get_cop1_config()

Gets the COP1 configuration for a data link.

Return type *Cop1Config*

get_cop1_status()

Retrieve the COP1 status.

Return type *Cop1Status*

get_info()

Get info on this link.

Return type *Link*

initialize_cop1 (*type*, *clcw_wait_timeout=None*, *v_r=None*)

Initialize COP1.

Parameters

- **type** (*str*) – One of WITH_CLCW_CHECK, WITHOUT_CLCW_CHECK, UNLOCK or SET_VR
- **clcw_wait_timeout** (*int*) – timeout in seconds used for the reception of CLCW. Required if type is WITH_CLCW_CHECK
- **v_r** (*int*) – value of v(R) if type is set to SET_VR

resume_cop1()

Resume COP1.

update_cop1_config (*window_width=None*, *timeout_type=None*, *tx_limit=None*, *tl=None*)

Sets the COP1 configuration for a data link.

Return type *Cop1Config*

class `yamcs.link.client.Cop1Subscription` (*manager*)

Bases: `yamcs.core.futures.WebSocketSubscriptionFuture`

Local object providing access to COP1 status updates.

Initializes the future. Should not be called by clients.

property `bypass_all`

property `cop1_active`

get_status()

Returns the latest known COP1 status.

Return type *Cop1Status*

property `nn_r`

property `state`

property `v_s`

5.1.2 Model

```
class yamcs.link.model.Cop1Config(proto)
    COPI configuration

    property t1
    property timeout_type
    property tx_limit
    property vc_id
        Virtual Channel ID.
    property window_width

class yamcs.link.model.Cop1Status(proto)
    COPI status

    property bypass_all
    property cop1_active
    property nn_r
    property state
    property v_s
```

5.2 Snippets

Create a *LinkClient* for a specific link:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
link = client.get_link(instance='simulator', link='udp-in')
```

Enable a link:

```
link.enable_link()
```

OBJECT STORAGE

The Storage API provides methods that you can use to programmatically work with Yamcs buckets and objects.

6.1 Reference

6.1.1 Client

class `yamcs.storage.StorageClient` (*ctx*, *instance='_global'*)

Client for working with buckets and objects managed by Yamcs.

create_bucket (*bucket_name*)

Create a new bucket.

Parameters **bucket_name** (*str*) – The name of the bucket.

download_object (*bucket_name*, *object_name*)

Download an object.

Parameters

- **bucket_name** (*str*) – The name of the bucket.
- **object_name** (*str*) – The object to fetch.

get_bucket (*name*)

Get a specific bucket.

Parameters **name** (*str*) – The bucket name.

Return type *Bucket*

list_buckets ()

List the buckets.

Return type *Iterable[Bucket]*

list_objects (*bucket_name*, *prefix=None*, *delimiter=None*)

List the objects for a bucket.

Parameters

- **bucket_name** (*str*) – The name of the bucket.
- **prefix** (*str*) – If specified, only objects that start with this prefix are listed.
- **delimiter** (*str*) – If specified, return only objects whose name do not contain the delimiter after the prefix. For the other objects, the response contains (in the prefix response parameter) the name truncated after the delimiter. Duplicates are omitted.

remove_bucket (*bucket_name*)

Remove a bucket.

Parameters **bucket_name** (*str*) – The name of the bucket.

remove_object (*bucket_name, object_name*)

Remove an object from a bucket.

Parameters

- **bucket_name** (*str*) – The name of the bucket.
- **object_name** (*str*) – The object to remove.

upload_object (*bucket_name, object_name, file_obj, content_type=None*)

Upload an object to a bucket.

Parameters

- **bucket_name** (*str*) – The name of the bucket.
- **object_name** (*str*) – The target name of the object.
- **file_obj** (*file*) – The file (or file-like object) to upload.
- **content_type** (*str*) – The content type associated to this object. This is mainly useful when accessing an object directly via a web browser. If unspecified, a content type *may* be automatically derived from the specified `file_obj`.

6.1.2 Model

class `yamcs.storage.model.Bucket` (*proto, storage_client*)

delete ()

Remove this bucket in its entirety.

delete_object (*object_name*)

Remove an object from this bucket.

Parameters **object_name** (*str*) – The object to remove.

download_object (*object_name*)

Download an object.

Parameters **object_name** (*str*) – The object to fetch.

list_objects (*prefix=None, delimiter=None*)

List the objects for this bucket.

Parameters

- **prefix** (*str*) – If specified, only objects that start with this prefix are listed.
- **delimiter** (*str*) – If specified, return only objects whose name do not contain the delimiter after the prefix. For the other objects, the response contains (in the `prefix` response parameter) the name truncated after the delimiter. Duplicates are omitted.

property name

Name of this bucket.

property object_count

Number of objects in this bucket.

property size

Total size in bytes of this bucket (excluding metadata).

upload_object (*object_name*, *file_obj*, *content_type=None*)

Upload an object to this bucket.

Parameters

- **object_name** (*str*) – The target name of the object.
- **file_obj** (*file*) – The file (or file-like object) to upload.
- **content_type** (*str*) – The content type associated to this object. This is mainly useful when accessing an object directly via a web browser. If unspecified, a content type *may* be automatically derived from the specified *file_obj*.

class `yamcs.storage.model.ObjectInfo` (*proto*, *bucket*, *storage_client*)

property created

Return when this object was created (or re-created).

Type `datetime`

delete ()

Remove this object.

download ()

Download this object.

property name

The name of this object.

property size

Size in bytes of this object (excluding metadata).

upload (*file_obj*)

Replace the content of this object.

Parameters **file_obj** (*file*) – The file (or file-like object) to upload.

class `yamcs.storage.model.ObjectListing` (*proto*, *bucket*, *storage_client*)

property objects

The objects in this listing.

Type `List[ObjectInfo]`

property prefixes

The prefixes in this listing.

Type `str[]`

6.2 Snippets

Create a *StorageClient* for a specific instance:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
storage = client.get_storage_client()
```


FILE TRANSFER

The File Transfer API provides methods that you can use to programmatically work with file transfers such as CFDP.

7.1 Reference

7.1.1 Client

class `yamcs.filetransfer.FileTransferClient` (*ctx, instance*)
Client for working with file transfers (e.g. CFDP) managed by Yamcs.

get_service (*name*)
Get a specific File Transfer service.

Parameters *name* (*str*) – The service name.

Return type *Service*

list_services ()
List the services.

Return type `Iterable[Service]`

7.1.2 Model

class `yamcs.filetransfer.model.Service` (*proto, service_client*)

cancel_transfer (*id*)
Cancel a transfer

create_transfer_subscription (*on_data=None, timeout=60*)
Create a new transfer subscription.

Parameters

- **on_data** – (Optional) Function that gets called with `TransferInfo` updates.
- **timeout** (*float*) – The amount of seconds to wait for the request to complete.

Returns Future that can be used to manage the background websocket subscription

Return type `TransferSubscription`

download (*bucket_name, object_name, remote_path, source_entity=None, destination_entity=None, overwrite=True, parents=True, reliable=False*)
Downloads a file from the destination to a bucket.

Parameters

- **bucket_name** (*str*) – Name of the bucket to receive the file.
- **object_name** (*str*) – Name of the file received in the bucket.
- **remote_path** (*str*) – Name of the file to be downloaded from the destination.
- **source_entity** (*str*) – Use a specific source entity. (useful in case of multiples)
- **destination_entity** (*str*) – Use a specific destination entity. (useful in case of multiples)
- **overwrite** (*bool*) – Replace file if it already exists.
- **parents** (*bool*) – Create the remote path if it does not yet exist.
- **reliable** (*bool*) – Enable reliable transfers.

Return type *Transfer*

property name

Name of this service.

pause_transfer (*id*)

Pauses a transfer

resume_transfer (*id*)

Resume a transfer

upload (*bucket_name, object_name, remote_path, source_entity=None, destination_entity=None, overwrite=True, parents=True, reliable=False*)

Uploads a file located in a bucket to a remote destination path.

Parameters

- **bucket_name** (*str*) – Name of the bucket containing the source object.
- **object_name** (*str*) – Name of the source object.
- **remote_path** (*str*) – Remote destination.
- **source_entity** (*str*) – Use a specific source entity. (useful in case of multiples)
- **destination_entity** (*str*) – Use a specific destination entity. (useful in case of multiples)
- **overwrite** (*bool*) – Replace a destination if it already exists.
- **parents** (*bool*) – Create the remote path if it does not yet exist.
- **reliable** (*bool*) – Whether to use a Class 2 CFDP transfer.

Return type *Transfer*

class `yamcs.filetransfer.model.Transfer` (*proto, service_client*)

Represents a file transfer.

await_complete (*timeout=None*)

Wait for the transfer to be completed.

Parameters **timeout** (*float*) – The amount of seconds to wait.

property bucket

cancel ()

Cancel this transfer

property error

Error message in case the transfer failed.

property id

Yamcs-local transfer identifier.

is_complete()

Returns whether this transfer is complete. A transfer can be completed, yet still failed.

is_success()

Returns true if this transfer was completed successfully.

property object_name**pause()**

Pause this transfer

property reliable

True if this is a Class 2 CFDP transfer.

property remote_path**resume()**

Resume this transfer

property size

Total bytes to transfer.

property state

Current transfer state.

property time

Time when the transfer was started.

property transferred_size

Total bytes already transferred.

7.2 Snippets

Create a *FileTransferClient* for a specific instance:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
cfdp = client.get_file_transfer_client(instance='cfdp')

# Operations are grouped by service.
# Here: take the first available
service = next(cfdp.list_services())
```


TIME CORRELATION (TCO)

The Time Correlation API provides methods that you can use to programmatically interact with a Yamcs TCO service.

8.1 Reference

8.1.1 Client

class `yamcs.tco.client.TCOClient` (*ctx, instance, service*)

Client for interacting with a Time Correlation service managed by Yamcs.

add_tof_interval (*start, stop, polynomial*)

Defines a ToF interval for the ERT range [*start, stop*], specifying a polynomial function of the form: $tof = a + bx + cx^2 + \dots$ where *x* is ERT minus the provided start date.

Parameters

- **start** (*datetime*) – ERT start
- **stop** (*datetime*) – ERT stop
- **polynomial** (*float []*) – Coefficients in the order [*a, b, c, ...*]

add_tof_intervals (*intervals*)

Adds multiple ToF intervals at once.

Parameters intervals (*ToFInterval []*) – List of ToF intervals.

get_status ()

Retrieve the TCO status.

Return type *TCOStatus*

override_coefficients (*utc, obt, gradient=0, offset=0*)

Manually override the association between UTC and onboard time.

Note: If later on you want to revert to automatically computed coefficients, use `reset_coefficients()`.

Parameters

- **utc** (*datetime*) – UTC
- **obt** (*int*) – Onboard time
- **gradient** (*Optional[float]*) – Gradient

- **offset** (*Optional[float]*) – Offset

reconfigure (*accuracy=None, validity=None, ob_delay=None, default_tof=None*)

Updates one or more TCO options

Parameters

- **accuracy** (*Optional[float]*) – Accuracy in seconds.
- **validity** (*Optional[float]*) – Validity in seconds.
- **ob_delay** (*Optional[float]*) – Onboard delay in seconds.
- **default_tof** (*Optional[float]*) – Default ToF in seconds. This value is used if the ToF estimator does not find a matching interval.

remove_tof_intervals (*start, stop*)

Removes previously registered ToF intervals whose start date falls in the specified range [*start, stop*].

Parameters

- **start** (*datetime*) – ERT start
- **stop** (*datetime*) – ERT stop

reset_coefficients ()

Resets current TCO coefficients, as well as any collected samples.

8.1.2 Model

class `yamcs.tco.model.TCOCoefficients` (*proto*)

TCO Coefficients

property `gradient`

property `obt`

property `offset`

property `utc`

class `yamcs.tco.model.TCOSample` (*proto*)

property `obt`

property `utc`

class `yamcs.tco.model.TCOStatus` (*proto*)

TCO Status

property `coefficients`

Current coefficients. Or None if the synchronization is not yet established.

property `coefficients_time`

Time when the coefficients have been computed

property `deviation`

Last computed deviation

property `samples`

The last accumulated samples

class `yamcs.tco.model.TofInterval` (*start, stop, polynomial*)

ToF interval for the ERT range [*start, stop*], specifying a polynomial function of the form: $tof = a + bx + cx^2 + \dots$ where x is ERT minus the provided start date.

Parameters

- **start** (*datetime*) – ERT start
- **stop** (*datetime*) – ERT stop
- **polynomial** (*float[]*) – Coefficients in the order [*a, b, c, ...*]

8.2 Snippets

Create a *TCOClient* for a specific instance:

```
from yamcs.client import YamcsClient

client = YamcsClient('localhost:8090')
tco = client.get_tco_client(instance='pus', service='tco0')
```


EXAMPLES

All of these examples run against a simulation that is used to develop and demo Yamcs. This setup is linked to a simple simulator which emits TM in the form of CCSDS packets and accepts a few telecommands as well.

Most of the telemetered parameters are in a space system called /YSS/SIMULATOR.

9.1 alarms.py

```
from time import sleep

from yamcs.client import YamcsClient

def receive_callbacks():
    """Registers an alarm callback."""

    def callback(alarm_update):
        print("Alarm Update:", alarm_update)

    processor.create_alarm_subscription(callback)

def acknowledge_all():
    """Acknowledges all active alarms."""
    for alarm in processor.list_alarms():
        if not alarm.is_acknowledged:
            processor.acknowledge_alarm(alarm, comment="false alarm")

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    processor = client.get_processor(instance="simulator", processor="realtime")
    receive_callbacks()
    sleep(10)

    print("Acknowledging all...")
    acknowledge_all()

    # If a parameter remains out of limits, a new alarm instance is created
    # on the next value update. So you would keep receiving callbacks on
    # the subscription.

    # The subscription is non-blocking. Prevent the main
```

(continues on next page)

(continued from previous page)

```
# thread from exiting
while True:
    sleep(10)
```

9.2 archive_breakdown.py

```
from yamcs.client import YamcsClient

def print_packet_count():
    """Print the number of packets grouped by packet name."""
    for name in archive.list_packet_names():
        packet_count = 0
        for group in archive.list_packet_histogram(name):
            for rec in group.records:
                packet_count += rec.count
        print(f" {name: <40} {packet_count: >20}")

def print_pp_groups():
    """Print the number of processed parameter frames by group name."""
    for group in archive.list_processed_parameter_groups():
        frame_count = 0
        for pp_group in archive.list_processed_parameter_group_histogram(group):
            for rec in pp_group.records:
                frame_count += rec.count
        print(f" {group: <40} {frame_count: >20}")

def print_event_count():
    """Print the number of events grouped by source."""
    for source in archive.list_event_sources():
        event_count = 0
        for group in archive.list_event_histogram(source):
            for rec in group.records:
                event_count += rec.count
        print(f" {source: <40} {event_count: >20}")

def print_command_count():
    """Print the number of commands grouped by name."""
    mdb = client.get_mdb(instance="simulator")
    for command in mdb.list_commands():
        total = 0
        for group in archive.list_command_histogram(command.qualified_name):
            for rec in group.records:
                total += rec.count
        print(f" {command.qualified_name: <40} {total: >20}")

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    archive = client.get_archive(instance="simulator")
```

(continues on next page)

(continued from previous page)

```

print("Packets:")
print_packet_count()

print("\nProcessed Parameter Groups:")
print_pp_groups()

print("\nEvents:")
print_event_count()

print("\nCommands:")
print_command_count()

```

9.3 archive_retrieval.py

```

from datetime import datetime, timedelta, timezone
from itertools import islice

from yamcs.client import YamcsClient

def print_last_packets():
    """Print the last 10 packets."""
    for packet in islice(archive.list_packets(descending=True), 0, 10):
        print(packet)

def print_packet_range():
    """Print the range of archived packets."""
    first_packet = next(iter(archive.list_packets()))
    last_packet = next(iter(archive.list_packets(descending=True)))
    print("First packet:", first_packet)
    print("Last packet:", last_packet)

    td = last_packet.generation_time - first_packet.generation_time
    print("Timespan:", td)

def iterate_specific_packet_range():
    """Count the number of packets in a specific range."""
    now = datetime.now(tz=timezone.utc)
    start = now - timedelta(hours=1)

    total = 0
    for packet in archive.list_packets(start=start, stop=now):
        total += 1
        # print(packet)
    print("Found", total, "packets in range")

def export_raw_packets():
    """Export raw packet binary."""
    now = datetime.now(tz=timezone.utc)
    start = now - timedelta(hours=1)

```

(continues on next page)

(continued from previous page)

```

with open("/tmp/dump.raw", "wb") as f:
    for chunk in archive.export_packets(start=start, stop=now):
        f.write(chunk)

def iterate_specific_event_range():
    """Count the number of events in a specific range."""
    now = datetime.now(tz=timezone.utc)
    start = now - timedelta(hours=1)

    total = 0
    for event in archive.list_events(start=start, stop=now):
        total += 1
        # print(event)
    print("Found", total, "events in range")

def print_last_values():
    """Print the last 10 values."""
    iterable = archive.list_parameter_values(
        "/YSS/SIMULATOR/BatteryVoltage1", descending=True
    )
    for pval in islice(iterable, 0, 10):
        print(pval)

def iterate_specific_parameter_range():
    """Count the number of parameter values in a specific range."""
    now = datetime.now(tz=timezone.utc)
    start = now - timedelta(hours=1)

    total = 0
    for pval in archive.list_parameter_values(
        "/YSS/SIMULATOR/BatteryVoltage1", start=start, stop=now
    ):
        total += 1
        # print(pval)
    print("Found", total, "parameter values in range")

def stream_specific_parameter_range():
    """Stream one or more parameters in a specific range."""
    now = datetime.now(tz=timezone.utc)
    start = now - timedelta(hours=1)

    total = 0
    for pdata in archive.stream_parameter_values(
        ["/YSS/SIMULATOR/BatteryVoltage1", "/YSS/SIMULATOR/BatteryVoltage2"],
        start=start,
        stop=now,
    ):
        total += 1
        # print(pdata)
    print("Found", total, "updates in range")

def print_last_commands():

```

(continues on next page)

(continued from previous page)

```

    """Print the last 10 commands."""
    iterable = archive.list_command_history(descending=True)
    for entry in islice(iterable, 0, 10):
        print(entry)

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    archive = client.get_archive(instance="simulator")

    print("Last 10 packets:")
    print_last_packets()

    print("\nPacket range:")
    print_packet_range()

    print("\nIterate specific packet range:")
    iterate_specific_packet_range()

    print("\nIterate specific event range:")
    iterate_specific_event_range()

    print("\nLast 10 parameter values:")
    print_last_values()

    print("\nIterate specific parameter range:")
    iterate_specific_parameter_range()

    # print("\nStream specific parameter range:")
    # stream_specific_parameter_range()

    print("\nLast 10 commands:")
    print_last_commands()

```

9.4 authenticate.py

```

from time import sleep

from yamcs.client import YamcsClient
from yamcs.core.auth import Credentials

# For this example to work, enable security in Yamcs by
# configuring appropriate authentication modules.

def authenticate_with_username_password():
    """Authenticate by directly providing username/password to Yamcs."""
    credentials = Credentials(username="admin", password="password")
    client = YamcsClient("localhost:8090", credentials=credentials)

    for link in client.list_links("simulator"):
        print(link)

```

(continues on next page)

(continued from previous page)

```

def authenticate_with_access_token(access_token):
    """Authenticate using an existing access token."""
    credentials = Credentials(access_token=access_token)
    client = YamcsClient("localhost:8090", credentials=credentials)

    for link in client.list_links("simulator"):
        print(link)

def impersonate_with_client_credentials():
    credentials = Credentials(
        client_id="cf79cfbd-ed01-4ae2-93e1-c606a2ebc36f",
        client_secret="!#?hgbul*3",
        become="admin",
    )
    client = YamcsClient("localhost:8090", credentials=credentials)
    print("have", client.get_user_info().username)
    while True:
        print(client.get_time("simulator"))
        sleep(1)

if __name__ == "__main__":
    print("Authenticate with username/password")
    authenticate_with_username_password()

    print("\nImpersonate with client credentials")
    impersonate_with_client_credentials()

```

9.5 ccscds_completeness.py

Note: CCSDS Completeness is a concept specific to CCSDS-style packets. If you store a different type of packet in Yamcs, then this code example is not applicable.

```

from datetime import datetime, timedelta, timezone

from yamcs.client import YamcsClient

def print_latest():
    """
    Prints completeness records for the last two days
    """
    now = datetime.now(tz=timezone.utc)
    start = now - timedelta(days=2)

    # Results are returned in records per 'group'.
    # A completeness group matches with a CCSDS APID.

    # We may receive multiple groups with the same APID
    # depending on how much data there is for the selected
    # range.

```

(continues on next page)

(continued from previous page)

```

# Combine all returned pages by APID
records_by_apid = {}
for group in archive.list_completeness_index(start=start, stop=now):
    if group.name not in records_by_apid:
        records_by_apid[group.name] = []
    records_by_apid[group.name].extend(group.records)

for apid, records in records_by_apid.items():
    print("APID:", apid)

    total = 0
    for rec in records:
        print("  ", rec)
        total += rec.count
    print(f" --> Total packets for {apid}: {total}")

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    archive = client.get_archive(instance="simulator")
    print_latest()

```

9.6 commanding.py

```

from time import sleep

from yamcs.client import YamcsClient
from yamcs.tmtc.model import VerificationConfig

def issue_command():
    """Issue a command to turn battery 1 off."""
    command = processor.issue_command(
        "/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF", args={"voltage_num": 1}
    )
    print("Issued", command)

def issue_command_modify_verification():
    """Issue a command with changed verification."""
    verification = VerificationConfig()
    verification.disable("Started")
    verification.modify_check_window("Queued", 1, 5)

    command = processor.issue_command(
        "/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF",
        args={"voltage_num": 1},
        verification=verification,
    )

    print("Issued", command)

```

(continues on next page)

(continued from previous page)

```

def issue_command_no_verification():
    """Issue a command with no verification."""
    verification = VerificationConfig()
    verification.disable()

    command = processor.issue_command(
        "/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF",
        args={"voltage_num": 1},
        verification=verification,
    )

    print("Issued", command)

def monitor_command():
    """Monitor command completion."""
    conn = processor.create_command_connection()

    command1 = conn.issue("/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF", args={"voltage_num": 1})
    ↪)

    # Issue 2nd command only if the previous command was completed successfully.
    command1.await_complete()
    if command1.is_success():
        conn.issue("/YSS/SIMULATOR/SWITCH_VOLTAGE_ON", args={"voltage_num": 1})
    else:
        print("Command 1 failed:", command1.error)

def monitor_acknowledgment():
    """Monitor command acknowledgment."""
    conn = processor.create_command_connection()

    command = conn.issue("/YSS/SIMULATOR/SWITCH_VOLTAGE_OFF", args={"voltage_num": 1})

    ack = command.await_acknowledgment("Acknowledge_Sent")
    print(ack.status)

def listen_to_command_history():
    """Receive updates on command history updates."""

    def tc_callback(rec):
        print("TC:", rec)

    processor.create_command_history_subscription(on_data=tc_callback)

def tm_callback(delivery):
    for parameter in delivery.parameters:
        print("TM:", parameter)

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    processor = client.get_processor("simulator", "realtime")

```

(continues on next page)

(continued from previous page)

```

print("Start to listen to command history")
listen_to_command_history()

print("Issue a command")
issue_command()

# Monitor the voltage parameter to confirm that it is 0
subscription = processor.create_parameter_subscription(
    ["/YSS/SIMULATOR/BatteryVoltage1"], on_data=tm_callback
)

# Subscription is non-blocking
sleep(20)

```

9.7 cop1.py

```

from time import sleep

from yamcs.client import YamcsClient

def callback(status):
    print("<callback> status:", status)

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    link = client.get_link("simulator", link="UDP_FRAME_OUT.vc0")

    config = link.get_cop1_config()
    print(config)

    print("Changing COP1 configuration")
    link.update_cop1_config(t1=3.1, tx_limit=4)

    monitor = link.create_cop1_subscription(on_data=callback)
    print("COP1 status subscribed.")

    sleep(5)

    print("Disabling COP1....")
    link.disable_cop1()

    sleep(3)
    print("Initializing COP1 with CLCW_CHECK")
    print(" (if no CLCW is received, COP1 will be suspended in 3 seconds)")
    link.initialize_cop1("WITH_CLCW_CHECK", clcw_wait_timeout=3)

    sleep(5)

    if monitor.state == "SUSPENDED":
        print("Resuming COP1")
        link.resume_cop1()

```

(continues on next page)

(continued from previous page)

```

sleep(3)

print("Disabling COP1...")
link.disable_cop1()

sleep(3)

print("Initializing COP1 with set v(R)=200")
print(" (if no CLCW is received, COP1 will be suspended in 3 seconds)")
link.initialize_cop1("SET_VR", v_r=200)

sleep(2)

```

9.8 file_transfer.py

```

import io

from yamcs.client import YamcsClient

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    storage = client.get_storage_client()
    cfdp = client.get_file_transfer_client(instance="cfdp")

    # Use pre-existing buckets, one for each direction
    out_bucket = storage.get_bucket("cfdpUp")
    in_bucket = storage.get_bucket("cfdpDown")

    # Prepare a sample file
    file_like = io.StringIO("Sample file content")
    out_bucket.upload_object("myfile", file_like)

    # Assume only one CFDP service
    service = next(cfdp.list_services())

    # Transfer myfile from bucket to spacecraft
    upload = service.upload(out_bucket.name, "myfile", "/CF:/mytarget")
    upload.await_complete(timeout=10)

    if not upload.is_success():
        print("Upload failure:", upload.error)
    else:
        print(f"Successfully uploaded {upload.remote_path} ({upload.size} bytes)")

    # Transfer myfile, but use an alternative destination entity
    upload = service.upload(
        out_bucket.name, "myfile", "/CF:/mytarget", destination_entity="target2",
    )
    upload.await_complete(timeout=10)

    if not upload.is_success():
        print("Upload failure:", upload.error)
    else:
        print(f"Successfully uploaded {upload.remote_path} ({upload.size} bytes)")

```

9.9 links.py

```

from time import sleep

from yamcs.client import YamcsClient

def enable_link(link):
    """Enable a link."""
    link.enable_link()

def callback(message):
    print("Link Event:", message)

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    link = client.get_link("simulator", link="tm_dump")

    print("Enabling link")
    enable_link(link)

    subscription = client.create_link_subscription("simulator", callback)

    sleep(10)

    print("-----")
    # You don't have to use the on_data callback. You could also
    # directly retrieve the latest data link state from a local cache:
    print("Last values from cache:")
    for link in subscription.list_links():
        print(link)

```

9.10 events.py

```

from time import sleep

from yamcs.client import YamcsClient

def listen_to_event_updates():
    """Subscribe to events."""

    def callback(event):
        print("Event:", event)

    client.create_event_subscription(instance="simulator", on_data=callback)

    sleep(5) # Subscription is non-blocking

def send_event():
    """Post an event."""

```

(continues on next page)

(continued from previous page)

```

client.send_event(instance="simulator", message="hello world")

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    listen_to_event_updates()

    print("Sending an event:")
    send_event()

    sleep(5)

```

9.11 mission_time.py

```

from time import sleep

from yamcs.client import YamcsClient

def callback(dt):
    print("Mission time:", dt)

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    subscription = client.create_time_subscription("simulator", callback)

    sleep(6)

    print("-----")
    # You don't have to use the on_data callback. You could also
    # directly retrieve the latest state from a local cache:
    print("Last time from cache:", subscription.time)

    # But then maybe you don't need a subscription, so do simply:
    time = client.get_time("simulator")
    print("Mission time (fresh from server)", time)

```

9.12 packet_subscription.py

```

from binascii import hexlify
from time import sleep

from yamcs.client import YamcsClient

def receive_callbacks():
    """Shows how to receive callbacks on packet updates."""

    def print_data(packet):
        hexpacket = hexlify(packet.binary).decode("ascii")

```

(continues on next page)

(continued from previous page)

```

    print(packet.generation_time, ":", hexpacket)

    processor.create_container_subscription(
        containers=["/YSS/SIMULATOR/FlightData", "/YSS/SIMULATOR/Power"],
        on_data=print_data,
    )

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    processor = client.get_processor("simulator", "realtime")

    print("\nReceive callbacks")
    receive_callbacks()

    sleep(5) # Subscription is non-blocking

```

9.13 parameter_subscription.py

```

from time import sleep

from yamcs.client import YamcsClient

def poll_values():
    """Shows how to poll values from the subscription."""
    subscription = processor.create_parameter_subscription(
        ["/YSS/SIMULATOR/BatteryVoltage1"]
    )

    sleep(5)
    print("Latest value:")
    print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage1"))

    sleep(5)
    print("Latest value:")
    print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage1"))

def receive_callbacks():
    """Shows how to receive callbacks on value updates."""

    def print_data(data):
        for parameter in data.parameters:
            print(parameter)

    processor.create_parameter_subscription(
        "/YSS/SIMULATOR/BatteryVoltage1", on_data=print_data
    )
    sleep(5) # Subscription is non-blocking

def manage_subscription():
    """Shows how to interact with a parameter subscription."""

```

(continues on next page)

(continued from previous page)

```
subscription = processor.create_parameter_subscription(
    ["/YSS/SIMULATOR/BatteryVoltage1"]
)

sleep(5)

print("Adding extra items to the existing subscription...")
subscription.add(
    [
        "/YSS/SIMULATOR/Alpha",
        "/YSS/SIMULATOR/BatteryVoltage2",
        "MDB:OPS Name/SIMULATOR_PrimBusVoltage1",
    ]
)

sleep(5)

print("Shrinking subscription...")
subscription.remove("/YSS/SIMULATOR/Alpha")

print("Cancelling the subscription...")
subscription.cancel()

print("Last values from cache:")
print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage1"))
print(subscription.get_value("/YSS/SIMULATOR/BatteryVoltage2"))
print(subscription.get_value("/YSS/SIMULATOR/Alpha"))
print(subscription.get_value("MDB:OPS Name/SIMULATOR_PrimBusVoltage1"))

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    processor = client.get_processor("simulator", "realtime")

    print("Poll value cache")
    poll_values()

    print("\nReceive callbacks")
    receive_callbacks()

    print("\nModify the subscription")
    manage_subscription()
```

9.14 plot_with_matplotlib.py

Note: To run this example, install matplotlib:

```
pip install matplotlib
```

```
from datetime import datetime, timedelta, timezone

import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

from yamcs.client import YamcsClient

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    archive = client.get_archive(instance="simulator")

    stop = datetime.now(tz=timezone.utc)
    start = stop - timedelta(hours=1)

    samples = archive.sample_parameter_values(
        "/YSS/SIMULATOR/Altitude", start=start, stop=stop
    )
    x = [s.time for s in samples]
    y = [s.avg for s in samples]
    plt.subplot(2, 1, 1)
    plt.title("Sampled at " + str(stop))
    plt.plot(x, y)
    plt.ylabel("Altitude")
    plt.grid()

    samples = archive.sample_parameter_values(
        "/YSS/SIMULATOR/SinkRate", start=start, stop=stop
    )
    x = [s.time for s in samples]
    y = [s.avg for s in samples]
    plt.subplot(2, 1, 2)
    plt.plot(x, y)
    plt.xlabel("UTC")
    plt.ylabel("Sink Rate")
    plt.grid()

    plt.gcf().canvas.set_window_title("Launch & Landing Simulator")
    plt.show()

```

9.15 query_mdb.py

```

from yamcs.client import YamcsClient

def print_space_systems():
    """Print all space systems."""
    for space_system in mdb.list_space_systems():
        print(space_system)

def print_parameters():
    """Print all float parameters."""
    for parameter in mdb.list_parameters(parameter_type="float"):
        print(parameter)

def print_commands():
    """Print all commands."""
    for command in mdb.list_commands():

```

(continues on next page)

(continued from previous page)

```

        print(command)

def find_parameter():
    """Find one parameter."""
    p1 = mdb.get_parameter("/YSS/SIMULATOR/BatteryVoltage2")
    print("Via qualified name:", p1)

    p2 = mdb.get_parameter("MDB:OPS Name/SIMULATOR_BatteryVoltage2")
    print("Via domain-specific alias:", p2)

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    mdb = client.get_mdb(instance="simulator")

    print("\nSpace systems:")
    print_space_systems()

    print("\nParameters:")
    print_parameters()

    print("\nCommands:")
    print_commands()

    print("\nFind a specific parameter using different names")
    find_parameter()

```

9.16 read_write_parameters.py

```

from yamcs.client import YamcsClient

def print_cached_value():
    """Print a single value from server cache."""
    pval = processor.get_parameter_value("/YSS/SIMULATOR/BatteryVoltage1")
    print(pval)

def print_realtime_value():
    """Print a newly processed value."""
    pval = processor.get_parameter_value(
        "/YSS/SIMULATOR/BatteryVoltage2", from_cache=False, timeout=5
    )
    print(pval)

def print_current_values():
    """Print multiple parameters from server cache."""
    pvals = processor.get_parameter_values(
        ["/YSS/SIMULATOR/BatteryVoltage1", "/YSS/SIMULATOR/BatteryVoltage2"]
    )
    print("battery1", pvals[0])
    print("battery2", pvals[1])

```

(continues on next page)

(continued from previous page)

```
def write_value():
    """Writes to a software parameter."""
    processor.set_parameter_value("/YSS/SIMULATOR/AllowCriticalTC1", True)

def write_values():
    """Writes multiple software parameters."""
    processor.set_parameter_values(
        {
            "/YSS/SIMULATOR/AllowCriticalTC1": False,
            "/YSS/SIMULATOR/AllowCriticalTC2": False,
        }
    )

if __name__ == "__main__":
    client = YamcsClient("localhost:8090")
    processor = client.get_processor(instance="simulator", processor="realtime")

    print("Fetch parameter value from cache")
    print_cached_value()

    print("\nFetch newly processed parameter value")
    print_realtime_value()

    print("\nFetch multiple parameters at the same time")
    print_current_values()

    print("\nWrite to a software parameter")
    write_value()

    print("\nWrite multiple software parameters at once")
    write_values()
```


PYTHON MODULE INDEX

y

yamcs.archive.model, 47
yamcs.core.auth, 11
yamcs.core.exceptions, 11
yamcs.core.futures, 12
yamcs.filetransfer.model, 61
yamcs.link.model, 55
yamcs.mdb.model, 17
yamcs.model, 7
yamcs.storage.model, 58
yamcs.tco.model, 66
yamcs.tmtc.model, 31

A

- abstract () (*yamcs.mdb.model.Command* property), 18
- access_token (*yamcs.core.auth.Credentials* attribute), 11
- acknowledge_alarm () (*yamcs.tmtc.client.ProcessorClient* method), 23
- acknowledge_message () (*yamcs.tmtc.model.Alarm* property), 31
- acknowledge_time () (*yamcs.tmtc.model.Alarm* property), 31
- acknowledged_by () (*yamcs.tmtc.model.Alarm* property), 31
- Acknowledgment (*class in yamcs.tmtc.model*), 31
- acknowledgments () (*yamcs.tmtc.model.CommandHistory* property), 32
- acknowledgments () (*yamcs.tmtc.model.MonitoredCommand* property), 34
- add () (*yamcs.tmtc.client.ParameterSubscription* method), 30
- add_done_callback () (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 12
- add_to_of_interval () (*yamcs.tco.client.TCOClient* method), 65
- add_to_of_intervals () (*yamcs.tco.client.TCOClient* method), 65
- Alarm (*class in yamcs.tmtc.model*), 31
- alarm () (*yamcs.tmtc.model.AlarmUpdate* property), 32
- AlarmSubscription (*class in yamcs.tmtc.client*), 29
- AlarmUpdate (*class in yamcs.tmtc.model*), 32
- Algorithm (*class in yamcs.mdb.model*), 17
- aliases () (*yamcs.mdb.model.Algorithm* property), 17
- aliases () (*yamcs.mdb.model.Command* property), 18
- aliases () (*yamcs.mdb.model.Container* property), 18
- aliases () (*yamcs.mdb.model.Parameter* property), 19
- aliases () (*yamcs.mdb.model.SpaceSystem* property), 20
- ArchiveClient (*class in yamcs.archive.client*), 41
- array_type () (*yamcs.mdb.model.ArrayType* property), 17
- array_type () (*yamcs.mdb.model.Member* property), 19
- array_type () (*yamcs.mdb.model.Parameter* property), 19
- ArrayType (*class in yamcs.mdb.model*), 17
- arrayType () (*yamcs.mdb.model.ArrayType* property), 17
- arrayType () (*yamcs.mdb.model.Member* property), 19
- arrayType () (*yamcs.mdb.model.Parameter* property), 19
- attributes () (*yamcs.tmtc.model.MonitoredCommand* property), 34
- AuthInfo (*class in yamcs.model*), 7
- avg () (*yamcs.archive.model.Sample* property), 48
- await_acknowledgment () (*yamcs.tmtc.model.MonitoredCommand* method), 34
- await_complete () (*yamcs.filetransfer.model.Transfer* method), 62
- await_complete () (*yamcs.tmtc.model.MonitoredCommand* method), 35

B

- base_command () (*yamcs.mdb.model.Command* property), 18
- become (*yamcs.core.auth.Credentials* attribute), 11
- before_request () (*yamcs.core.auth.Credentials* method), 11
- binary () (*yamcs.tmtc.model.CommandHistory* property), 32
- binary () (*yamcs.tmtc.model.ContainerData* property), 33
- binary () (*yamcs.tmtc.model.IssuedCommand* property), 34
- binary () (*yamcs.tmtc.model.Packet* property), 35
- bitlength () (*yamcs.mdb.model.DataEncoding* property), 18
- Bucket (*class in yamcs.storage.model*), 58
- bucket () (*yamcs.filetransfer.model.Transfer* property), 62

bypass_all() (*yamcs.link.client.Cop1Subscription property*), 54
 bypass_all() (*yamcs.link.model.Cop1Status property*), 55

C

Calibrator (*class in yamcs.tmtc.model*), 32
 cancel() (*yamcs.core.futures.WebSocketSubscriptionFuture method*), 12
 cancel() (*yamcs.filetransfer.model.Transfer method*), 62
 cancel_transfer() (*yamcs.filetransfer.model.Service method*), 61
 cancelled() (*yamcs.core.futures.WebSocketSubscriptionFuture method*), 12
 class_name() (*yamcs.model.Link property*), 8
 class_name() (*yamcs.model.Service property*), 10
 clear_alarm() (*yamcs.tmtc.client.ProcessorClient method*), 23
 clear_alarm_ranges() (*yamcs.tmtc.client.ProcessorClient method*), 23
 clear_cache() (*yamcs.tmtc.client.CommandHistorySubscription method*), 30
 clear_calibrators() (*yamcs.tmtc.client.ProcessorClient method*), 23
 client_id (*yamcs.core.auth.Credentials attribute*), 11
 client_secret (*yamcs.core.auth.Credentials attribute*), 11
 coefficients() (*yamcs.tco.model.TCOStatus property*), 66
 coefficients_time() (*yamcs.tco.model.TCOStatus property*), 66
 column_types() (*yamcs.archive.model.ResultSet property*), 48
 ColumnData (*class in yamcs.archive.model*), 47
 columns() (*yamcs.archive.model.ResultSet property*), 48
 columns() (*yamcs.archive.model.StreamData property*), 49
 Command (*class in yamcs.mdb.model*), 18
 CommandConnection (*class in yamcs.tmtc.client*), 29
 CommandHistory (*class in yamcs.tmtc.model*), 32
 CommandHistorySubscription (*class in yamcs.tmtc.client*), 30
 comment() (*yamcs.tmtc.model.CommandHistory property*), 33
 comment() (*yamcs.tmtc.model.MonitoredCommand property*), 35
 ConnectionFailure, 11
 consequence_level() (*yamcs.mdb.model.Significance property*), 20
 Container (*class in yamcs.mdb.model*), 18
 ContainerData (*class in yamcs.tmtc.model*), 33
 cop1_active() (*yamcs.link.client.Cop1Subscription property*), 54
 cop1_active() (*yamcs.link.model.Cop1Status property*), 55
 Cop1Config (*class in yamcs.link.model*), 55
 Cop1Status (*class in yamcs.link.model*), 55
 Cop1Subscription (*class in yamcs.link.client*), 54
 count() (*yamcs.archive.model.IndexRecord property*), 47
 count() (*yamcs.tmtc.model.Alarm property*), 31
 create_alarm_subscription() (*yamcs.tmtc.client.ProcessorClient method*), 24
 create_bucket() (*yamcs.storage.StorageClient method*), 57
 create_command_connection() (*yamcs.tmtc.client.ProcessorClient method*), 24
 create_command_history_subscription() (*yamcs.tmtc.client.ProcessorClient method*), 24
 create_container_subscription() (*yamcs.tmtc.client.ProcessorClient method*), 24
 create_cop1_subscription() (*yamcs.link.client.LinkClient method*), 53
 create_event_subscription() (*yamcs.client.YamcsClient method*), 3
 create_instance() (*yamcs.client.YamcsClient method*), 3
 create_link_subscription() (*yamcs.client.YamcsClient method*), 4
 create_packet_subscription() (*yamcs.tmtc.client.ProcessorClient method*), 24
 create_parameter_subscription() (*yamcs.tmtc.client.ProcessorClient method*), 25
 create_stream_subscription() (*yamcs.archive.client.ArchiveClient method*), 41
 create_time_subscription() (*yamcs.client.YamcsClient method*), 4
 create_transfer_subscription() (*yamcs.filetransfer.model.Service method*), 61
 created() (*yamcs.storage.model.ObjectInfo property*), 59
 Credentials (*class in yamcs.core.auth*), 11
 current_event() (*yamcs.tmtc.model.EventAlarm property*), 33
 current_value() (*yamcs.tmtc.model.ParameterAlarm property*), 35

D

data_encoding() (*yamcs.mdb.model.Parameter property*), 19
 data_source() (*yamcs.mdb.model.Parameter property*), 20

- DataEncoding (class in *yamcs.mdb.model*), 18
- default_yamcs_instance() (*yamcs.model.ServerInfo* property), 9
- delete() (*yamcs.storage.model.Bucket* method), 58
- delete() (*yamcs.storage.model.ObjectInfo* method), 59
- delete_object() (*yamcs.storage.model.Bucket* method), 58
- delivery_count (*yamcs.tmtc.client.ParameterSubscription* attribute), 30
- description() (*yamcs.mdb.model.Algorithm* property), 17
- description() (*yamcs.mdb.model.Command* property), 18
- description() (*yamcs.mdb.model.Container* property), 18
- description() (*yamcs.mdb.model.EnumValue* property), 19
- description() (*yamcs.mdb.model.Parameter* property), 20
- description() (*yamcs.mdb.model.SpaceSystem* property), 20
- deviation() (*yamcs.tco.model.TCOStatus* property), 66
- dimensions() (*yamcs.mdb.model.ArrayType* property), 18
- disable() (*yamcs.tmtc.model.VerificationConfig* method), 37
- disable_copl() (*yamcs.link.client.LinkClient* method), 53
- disable_link() (*yamcs.link.client.LinkClient* method), 53
- done() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 12
- download() (*yamcs.filetransfer.model.Service* method), 61
- download() (*yamcs.storage.model.ObjectInfo* method), 59
- download_object() (*yamcs.storage.model.Bucket* method), 58
- download_object() (*yamcs.storage.StorageClient* method), 57
- dump_table() (*yamcs.archive.client.ArchiveClient* method), 41
- ## E
- enable_link() (*yamcs.link.client.LinkClient* method), 53
- enabled() (*yamcs.model.Link* property), 8
- encoding() (*yamcs.mdb.model.DataEncoding* property), 18
- eng_value() (*yamcs.archive.model.ParameterRange* property), 48
- eng_value() (*yamcs.archive.model.ParameterRangeEntry* property), 48
- eng_value() (*yamcs.tmtc.model.ParameterValue* property), 36
- entries() (*yamcs.archive.model.ParameterRange* property), 48
- enum_values() (*yamcs.mdb.model.Parameter* property), 20
- EnumValue (class in *yamcs.mdb.model*), 19
- error() (*yamcs.filetransfer.model.Transfer* property), 62
- error() (*yamcs.tmtc.model.CommandHistory* property), 33
- error() (*yamcs.tmtc.model.MonitoredCommand* property), 35
- Event (class in *yamcs.model*), 7
- event_type() (*yamcs.model.Event* property), 7
- event_type() (*yamcs.model.LinkEvent* property), 9
- EventAlarm (class in *yamcs.tmtc.model*), 33
- exception() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 12
- execute_sql() (*yamcs.archive.client.ArchiveClient* method), 41
- expiry (*yamcs.core.auth.Credentials* attribute), 11
- export_packets() (*yamcs.archive.client.ArchiveClient* method), 41
- ## F
- failure_cause() (*yamcs.model.Instance* property), 8
- FileTransferClient (class in *yamcs.filetransfer*), 61
- ## G
- generation_time (*yamcs.tmtc.model.CommandHistory* attribute), 33
- generation_time() (*yamcs.model.Event* property), 8
- generation_time() (*yamcs.tmtc.model.ContainerData* property), 33
- generation_time() (*yamcs.tmtc.model.IssuedCommand* property), 34
- generation_time() (*yamcs.tmtc.model.Packet* property), 35
- generation_time() (*yamcs.tmtc.model.ParameterValue* property), 36
- get_alarm() (*yamcs.tmtc.client.AlarmSubscription* method), 29
- get_algorithm() (*yamcs.mdb.client.MDBClient* method), 16
- get_archive() (*yamcs.client.YamcsClient* method), 4

[get_auth_info\(\)](#) (*yamcs.client.YamcsClient method*), 4
[get_bucket\(\)](#) (*yamcs.storage.StorageClient method*), 57
[get_command\(\)](#) (*yamcs.mdb.client.MDBClient method*), 16
[get_command_history\(\)](#) (*yamcs.tmtc.client.CommandHistorySubscription method*), 30
[get_container\(\)](#) (*yamcs.mdb.client.MDBClient method*), 16
[get_cop1_config\(\)](#) (*yamcs.link.client.LinkClient method*), 54
[get_cop1_status\(\)](#) (*yamcs.link.client.LinkClient method*), 54
[get_file_transfer_client\(\)](#) (*yamcs.client.YamcsClient method*), 4
[get_info\(\)](#) (*yamcs.link.client.LinkClient method*), 54
[get_link\(\)](#) (*yamcs.client.LinkSubscription method*), 7
[get_link\(\)](#) (*yamcs.client.YamcsClient method*), 4
[get_mdb\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_packet\(\)](#) (*yamcs.archive.client.ArchiveClient method*), 42
[get_parameter\(\)](#) (*yamcs.mdb.client.MDBClient method*), 16
[get_parameter_value\(\)](#) (*yamcs.tmtc.client.ProcessorClient method*), 25
[get_parameter_values\(\)](#) (*yamcs.tmtc.client.ProcessorClient method*), 25
[get_processor\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_server_info\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_service\(\)](#) (*yamcs.filetransfer.FileTransferClient method*), 61
[get_space_system\(\)](#) (*yamcs.mdb.client.MDBClient method*), 16
[get_status\(\)](#) (*yamcs.link.client.Cop1Subscription method*), 54
[get_status\(\)](#) (*yamcs.tco.client.TCOClient method*), 65
[get_storage_client\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_stream\(\)](#) (*yamcs.archive.client.ArchiveClient method*), 42
[get_table\(\)](#) (*yamcs.archive.client.ArchiveClient method*), 42
[get_tco_client\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_time\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_user_info\(\)](#) (*yamcs.client.YamcsClient method*), 5
[get_value\(\)](#) (*yamcs.tmtc.client.ParameterSubscription method*), 30
[get_value\(\)](#) (*yamcs.tmtc.model.ParameterData method*), 36
[gradient\(\)](#) (*yamcs.tco.model.TCOCoefficients property*), 66

H

[hex\(\)](#) (*yamcs.tmtc.model.IssuedCommand property*), 34

I

[id\(\)](#) (*yamcs.filetransfer.model.Transfer property*), 63
[id\(\)](#) (*yamcs.model.ServerInfo property*), 9
[id\(\)](#) (*yamcs.tmtc.model.IssuedCommand property*), 34
[in_count\(\)](#) (*yamcs.model.Link property*), 8
[IndexGroup](#) (*class in yamcs.archive.model*), 47
[IndexRecord](#) (*class in yamcs.archive.model*), 47
[initialize_cop1\(\)](#) (*yamcs.link.client.LinkClient method*), 54
[Instance](#) (*class in yamcs.model*), 8
[instance\(\)](#) (*yamcs.model.Link property*), 8
[instance\(\)](#) (*yamcs.model.Processor property*), 9
[instance\(\)](#) (*yamcs.model.Service property*), 10
[InstanceTemplate](#) (*class in yamcs.model*), 8
[is_acknowledged\(\)](#) (*yamcs.tmtc.model.Alarm property*), 31
[is_complete\(\)](#) (*yamcs.filetransfer.model.Transfer method*), 63
[is_complete\(\)](#) (*yamcs.tmtc.model.CommandHistory method*), 33
[is_complete\(\)](#) (*yamcs.tmtc.model.MonitoredCommand method*), 35
[is_expired\(\)](#) (*yamcs.core.auth.Credentials method*), 11
[is_failure\(\)](#) (*yamcs.tmtc.model.CommandHistory method*), 33
[is_latched\(\)](#) (*yamcs.tmtc.model.Alarm property*), 31
[is_latching\(\)](#) (*yamcs.tmtc.model.Alarm property*), 31
[is_ok\(\)](#) (*yamcs.tmtc.model.Alarm property*), 31
[is_process_ok\(\)](#) (*yamcs.tmtc.model.Alarm property*), 31
[is_shelved\(\)](#) (*yamcs.tmtc.model.Alarm property*), 31
[is_success\(\)](#) (*yamcs.filetransfer.model.Transfer method*), 63
[is_success\(\)](#) (*yamcs.tmtc.model.CommandHistory method*), 33
[is_success\(\)](#) (*yamcs.tmtc.model.MonitoredCommand method*), 35
[is_terminated\(\)](#) (*yamcs.tmtc.model.Acknowledgment method*), 31
[issue\(\)](#) (*yamcs.tmtc.client.CommandConnection method*), 29

- issue_command() (*yamcs.tmtc.client.ProcessorClient method*), 26
 IssuedCommand (*class in yamcs.tmtc.model*), 34
- ## L
- label() (*yamcs.mdb.model.EnumValue property*), 19
 Link (*class in yamcs.model*), 8
 link() (*yamcs.model.LinkEvent property*), 9
 LinkClient (*class in yamcs.link.client*), 53
 LinkEvent (*class in yamcs.model*), 9
 LinkSubscription (*class in yamcs.client*), 7
 list_alarms() (*yamcs.tmtc.client.AlarmSubscription method*), 29
 list_alarms() (*yamcs.tmtc.client.ProcessorClient method*), 26
 list_algorithms() (*yamcs.mdb.client.MDBClient method*), 16
 list_buckets() (*yamcs.storage.StorageClient method*), 57
 list_command_histogram() (*yamcs.archive.client.ArchiveClient method*), 42
 list_command_history() (*yamcs.archive.client.ArchiveClient method*), 42
 list_commands() (*yamcs.mdb.client.MDBClient method*), 16
 list_completeness_index() (*yamcs.archive.client.ArchiveClient method*), 43
 list_containers() (*yamcs.mdb.client.MDBClient method*), 17
 list_data_links() (*yamcs.client.YamcsClient method*), 5
 list_event_histogram() (*yamcs.archive.client.ArchiveClient method*), 43
 list_event_sources() (*yamcs.archive.client.ArchiveClient method*), 43
 list_events() (*yamcs.archive.client.ArchiveClient method*), 43
 list_instance_templates() (*yamcs.client.YamcsClient method*), 5
 list_instances() (*yamcs.client.YamcsClient method*), 5
 list_links() (*yamcs.client.LinkSubscription method*), 7
 list_links() (*yamcs.client.YamcsClient method*), 5
 list_objects() (*yamcs.storage.model.Bucket method*), 58
 list_objects() (*yamcs.storage.StorageClient method*), 57
 list_packet_histogram() (*yamcs.archive.client.ArchiveClient method*), 43
 list_packet_names() (*yamcs.archive.client.ArchiveClient method*), 43
 list_packets() (*yamcs.archive.client.ArchiveClient method*), 44
 list_parameter_ranges() (*yamcs.archive.client.ArchiveClient method*), 44
 list_parameter_values() (*yamcs.archive.client.ArchiveClient method*), 45
 list_parameters() (*yamcs.mdb.client.MDBClient method*), 17
 list_processed_parameter_group_histogram() (*yamcs.archive.client.ArchiveClient method*), 45
 list_processed_parameter_groups() (*yamcs.archive.client.ArchiveClient method*), 45
 list_processors() (*yamcs.client.YamcsClient method*), 6
 list_services() (*yamcs.client.YamcsClient method*), 6
 list_services() (*yamcs.filetransfer.FileTransferClient method*), 61
 list_space_systems() (*yamcs.mdb.client.MDBClient method*), 17
 list_streams() (*yamcs.archive.client.ArchiveClient method*), 45
 list_tables() (*yamcs.archive.client.ArchiveClient method*), 45
 little_endian() (*yamcs.mdb.model.DataEncoding property*), 19
 load_table() (*yamcs.archive.client.ArchiveClient method*), 46
 login() (*yamcs.core.auth.Credentials method*), 11
 long_description() (*yamcs.mdb.model.Algorithm property*), 17
 long_description() (*yamcs.mdb.model.Command property*), 18
 long_description() (*yamcs.mdb.model.Container property*), 18
 long_description() (*yamcs.mdb.model.Parameter property*), 20
 long_description() (*yamcs.mdb.model.SpaceSystem property*), 20
- ## M
- max() (*yamcs.archive.model.Sample property*), 48
 MDBClient (*class in yamcs.mdb.client*), 16

Member (*class in yamcs.mdb.model*), 19
members () (*yamcs.mdb.model.ArrayType property*), 18
members () (*yamcs.mdb.model.Member property*), 19
members () (*yamcs.mdb.model.Parameter property*), 20
message (*yamcs.tmtc.model.Acknowledgment attribute*), 31
message () (*yamcs.model.Event property*), 8
min () (*yamcs.archive.model.Sample property*), 49
mission_time () (*yamcs.model.Instance property*), 8
mission_time () (*yamcs.model.Processor property*), 9
modify_check_window ()
 (*yamcs.tmtc.model.VerificationConfig method*), 37
MonitoredCommand (*class in yamcs.tmtc.model*), 34
monitoring_result ()
 (*yamcs.tmtc.model.ParameterValue property*), 36
most_severe_event ()
 (*yamcs.tmtc.model.EventAlarm property*), 33
most_severe_value ()
 (*yamcs.tmtc.model.ParameterAlarm property*), 35

N

name (*yamcs.tmtc.model.Acknowledgment attribute*), 31
name () (*yamcs.archive.model.ColumnData property*), 47
name () (*yamcs.archive.model.IndexGroup property*), 47
name () (*yamcs.archive.model.Stream property*), 49
name () (*yamcs.archive.model.Table property*), 49
name () (*yamcs.filetransfer.model.Service property*), 62
name () (*yamcs.mdb.model.Algorithm property*), 17
name () (*yamcs.mdb.model.ArrayType property*), 18
name () (*yamcs.mdb.model.Command property*), 18
name () (*yamcs.mdb.model.Container property*), 18
name () (*yamcs.mdb.model.Member property*), 19
name () (*yamcs.mdb.model.Parameter property*), 20
name () (*yamcs.mdb.model.SpaceSystem property*), 21
name () (*yamcs.model.Instance property*), 8
name () (*yamcs.model.InstanceTemplate property*), 8
name () (*yamcs.model.Link property*), 8
name () (*yamcs.model.ObjectPrivilege property*), 9
name () (*yamcs.model.Processor property*), 9
name () (*yamcs.model.Service property*), 10
name () (*yamcs.storage.model.Bucket property*), 58
name () (*yamcs.storage.model.ObjectInfo property*), 59
name () (*yamcs.tmtc.model.Alarm property*), 32
name () (*yamcs.tmtc.model.CommandHistory property*), 33
name () (*yamcs.tmtc.model.ContainerData property*), 33
name () (*yamcs.tmtc.model.IssuedCommand property*), 34

name () (*yamcs.tmtc.model.Packet property*), 35
name () (*yamcs.tmtc.model.ParameterValue property*), 36
nn_r () (*yamcs.link.client.Cop1Subscription property*), 54
nn_r () (*yamcs.link.model.Cop1Status property*), 55
NotFound, 11

O

object_count () (*yamcs.storage.model.Bucket property*), 58
object_name () (*yamcs.filetransfer.model.Transfer property*), 63
object_privileges () (*yamcs.model.UserInfo property*), 10
ObjectInfo (*class in yamcs.storage.model*), 59
ObjectListing (*class in yamcs.storage.model*), 59
ObjectPrivilege (*class in yamcs.model*), 9
objects () (*yamcs.model.ObjectPrivilege property*), 9
objects () (*yamcs.storage.model.ObjectListing property*), 59
obt () (*yamcs.tco.model.TCOCoefficients property*), 66
obt () (*yamcs.tco.model.TCOSample property*), 66
offset () (*yamcs.tco.model.TCOCoefficients property*), 66
origin () (*yamcs.tmtc.model.CommandHistory property*), 33
origin () (*yamcs.tmtc.model.IssuedCommand property*), 34
out_count () (*yamcs.model.Link property*), 9
override_coefficients ()
 (*yamcs.tco.client.TCOClient method*), 65
owner () (*yamcs.model.Processor property*), 9

P

Packet (*class in yamcs.tmtc.model*), 35
Parameter (*class in yamcs.mdb.model*), 19
parameter_count ()
 (*yamcs.archive.model.ParameterRange property*), 48
parameter_count ()
 (*yamcs.archive.model.ParameterRangeEntry property*), 48
parameter_count () (*yamcs.archive.model.Sample property*), 49
ParameterAlarm (*class in yamcs.tmtc.model*), 35
ParameterData (*class in yamcs.tmtc.model*), 35
ParameterRange (*class in yamcs.archive.model*), 48
ParameterRangeEntry (*class in yamcs.archive.model*), 48
parameters () (*yamcs.tmtc.model.ParameterData property*), 36
ParameterSubscription (*class in yamcs.tmtc.client*), 30

- ParameterValue (class in *yamcs.tmtc.model*), 36
 password (*yamcs.core.auth.Credentials* attribute), 11
 pause() (*yamcs.filetransfer.model.Transfer* method), 63
 pause_transfer() (*yamcs.filetransfer.model.Service* method), 62
 persistent() (*yamcs.model.Processor* property), 9
 POLYNOMIAL (*yamcs.tmtc.model.Calibrator* attribute), 32
 prefixes() (*yamcs.storage.model.ObjectListing* property), 59
 processing_status() (*yamcs.tmtc.model.ParameterValue* property), 36
 Processor (class in *yamcs.model*), 9
 processor() (*yamcs.model.Service* property), 10
 ProcessorClient (class in *yamcs.tmtc.client*), 23
- ## Q
- qualified_name() (*yamcs.mdb.model.Algorithm* property), 17
 qualified_name() (*yamcs.mdb.model.Command* property), 18
 qualified_name() (*yamcs.mdb.model.Container* property), 18
 qualified_name() (*yamcs.mdb.model.Parameter* property), 20
 qualified_name() (*yamcs.mdb.model.SpaceSystem* property), 21
 queue() (*yamcs.tmtc.model.IssuedCommand* property), 34
- ## R
- range_condition() (*yamcs.tmtc.model.ParameterValue* property), 36
 RangeSet (class in *yamcs.tmtc.model*), 36
 raw_value() (*yamcs.tmtc.model.ParameterValue* property), 36
 reason() (*yamcs.mdb.model.Significance* property), 20
 rebuild_histogram() (*yamcs.archive.client.ArchiveClient* method), 46
 rebuild_parameter_archive() (*yamcs.archive.client.ArchiveClient* method), 46
 reception_time() (*yamcs.model.Event* property), 8
 reception_time() (*yamcs.tmtc.model.ContainerData* property), 33
 reception_time() (*yamcs.tmtc.model.Packet* property), 35
 reception_time() (*yamcs.tmtc.model.ParameterValue* property), 36
 reconfigure() (*yamcs.tco.client.TCOClient* method), 66
 records() (*yamcs.archive.model.IndexGroup* property), 47
 refresh() (*yamcs.core.auth.Credentials* method), 11
 refresh_token (*yamcs.core.auth.Credentials* attribute), 11
 reliable() (*yamcs.filetransfer.model.Transfer* property), 63
 remote_path() (*yamcs.filetransfer.model.Transfer* property), 63
 remove() (*yamcs.tmtc.client.ParameterSubscription* method), 31
 remove_bucket() (*yamcs.storage.StorageClient* method), 57
 remove_object() (*yamcs.storage.StorageClient* method), 58
 remove_tof_intervals() (*yamcs.tco.client.TCOClient* method), 66
 reply() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 12
 require_authentication() (*yamcs.model.AuthInfo* property), 7
 reset_alarm_ranges() (*yamcs.tmtc.client.ProcessorClient* method), 26
 reset_algorithm() (*yamcs.tmtc.client.ProcessorClient* method), 26
 reset_calibrators() (*yamcs.tmtc.client.ProcessorClient* method), 26
 reset_coefficients() (*yamcs.tco.client.TCOClient* method), 66
 restart_instance() (*yamcs.client.YamcsClient* method), 6
 result() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 12
 ResultSet (class in *yamcs.archive.model*), 48
 resume() (*yamcs.filetransfer.model.Transfer* method), 63
 resume_cop1() (*yamcs.link.client.LinkClient* method), 54
 resume_transfer() (*yamcs.filetransfer.model.Service* method), 62
 running() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 13
- ## S
- Sample (class in *yamcs.archive.model*), 48
 sample_parameter_values() (*yamcs.archive.client.ArchiveClient* method), 46
 samples() (*yamcs.tco.model.TCOStatus* property), 66
 send_event() (*yamcs.client.YamcsClient* method), 6

- sequence_number() (*yamcs.model.Event* property), 8
 - sequence_number() (*yamcs.tmtc.model.Alarm* property), 32
 - sequence_number() (*yamcs.tmtc.model.CommandHistory* property), 33
 - sequence_number() (*yamcs.tmtc.model.IssuedCommand* property), 34
 - sequence_number() (*yamcs.tmtc.model.Packet* property), 35
 - ServerInfo (*class in yamcs.model*), 9
 - Service (*class in yamcs.filetransfer.model*), 61
 - Service (*class in yamcs.model*), 10
 - set_alarm_range_sets() (*yamcs.tmtc.client.ProcessorClient* method), 27
 - set_algorithm() (*yamcs.tmtc.client.ProcessorClient* method), 27
 - set_calibrators() (*yamcs.tmtc.client.ProcessorClient* method), 27
 - set_default_alarm_ranges() (*yamcs.tmtc.client.ProcessorClient* method), 27
 - set_default_calibrator() (*yamcs.tmtc.client.ProcessorClient* method), 28
 - set_exception() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 13
 - set_parameter_value() (*yamcs.tmtc.client.ProcessorClient* method), 28
 - set_parameter_values() (*yamcs.tmtc.client.ProcessorClient* method), 28
 - set_result() (*yamcs.core.futures.WebSocketSubscriptionFuture* method), 13
 - severity() (*yamcs.model.Event* property), 8
 - severity() (*yamcs.tmtc.model.Alarm* property), 32
 - shelve_alarm() (*yamcs.tmtc.client.ProcessorClient* method), 28
 - Significance (*class in yamcs.mdb.model*), 20
 - significance() (*yamcs.mdb.model.Command* property), 18
 - size() (*yamcs.filetransfer.model.Transfer* property), 63
 - size() (*yamcs.storage.model.Bucket* property), 58
 - size() (*yamcs.storage.model.ObjectInfo* property), 59
 - source() (*yamcs.model.Event* property), 8
 - source() (*yamcs.tmtc.model.CommandHistory* property), 33
 - source() (*yamcs.tmtc.model.IssuedCommand* property), 34
 - SpaceSystem (*class in yamcs.mdb.model*), 20
 - SPLINE (*yamcs.tmtc.model.Calibrator* attribute), 32
 - start() (*yamcs.archive.model.IndexRecord* property), 47
 - start() (*yamcs.archive.model.ParameterRange* property), 48
 - start_instance() (*yamcs.client.YamcsClient* method), 6
 - start_service() (*yamcs.client.YamcsClient* method), 6
 - state() (*yamcs.filetransfer.model.Transfer* property), 63
 - state() (*yamcs.link.client.Cop1Subscription* property), 54
 - state() (*yamcs.link.model.Cop1Status* property), 55
 - state() (*yamcs.model.Instance* property), 8
 - state() (*yamcs.model.Processor* property), 9
 - state() (*yamcs.model.Service* property), 10
 - status (*yamcs.tmtc.model.Acknowledgment* attribute), 31
 - status() (*yamcs.model.Link* property), 9
 - stop() (*yamcs.archive.model.IndexRecord* property), 48
 - stop() (*yamcs.archive.model.ParameterRange* property), 48
 - stop_instance() (*yamcs.client.YamcsClient* method), 7
 - stop_service() (*yamcs.client.YamcsClient* method), 7
 - StorageClient (*class in yamcs.storage*), 57
 - Stream (*class in yamcs.archive.model*), 49
 - stream_data() (*yamcs.archive.model.StreamData* property), 49
 - stream_parameter_values() (*yamcs.archive.client.ArchiveClient* method), 47
 - StreamData (*class in yamcs.archive.model*), 49
 - streamer() (*yamcs.model.UserInfo* property), 10
 - system_privileges() (*yamcs.model.UserInfo* property), 10
- ## T
- t1() (*yamcs.link.model.Cop1Config* property), 55
 - Table (*class in yamcs.archive.model*), 49
 - TCOClient (*class in yamcs.tco.client*), 65
 - TCOCoefficients (*class in yamcs.tco.model*), 66
 - TCOSample (*class in yamcs.tco.model*), 66
 - TCOStatus (*class in yamcs.tco.model*), 66
 - time (*yamcs.client.TimeSubscription* attribute), 7
 - time (*yamcs.tmtc.model.Acknowledgment* attribute), 31
 - time() (*yamcs.archive.model.Sample* property), 49
 - time() (*yamcs.filetransfer.model.Transfer* property), 63
 - timeout_type() (*yamcs.link.model.Cop1Config* property), 55
 - TimeoutError, 11
 - TimeSubscription (*class in yamcs.client*), 7
 - TofInterval (*class in yamcs.tco.model*), 66
 - Transfer (*class in yamcs.filetransfer.model*), 62
 - transferred_size() (*yamcs.filetransfer.model.Transfer* property),

- 63
- `trigger_event()` (*yamcs.tmtc.model.EventAlarm property*), 34
- `trigger_time()` (*yamcs.tmtc.model.Alarm property*), 32
- `trigger_value()` (*yamcs.tmtc.model.ParameterAlarm property*), 35
- `tx_limit()` (*yamcs.link.model.Cop1Config property*), 55
- `type()` (*yamcs.mdb.model.DataEncoding property*), 19
- `type()` (*yamcs.mdb.model.Member property*), 19
- `type()` (*yamcs.mdb.model.Parameter property*), 20
- `type()` (*yamcs.model.Processor property*), 9
- ## U
- Unauthorized, 12
- `units()` (*yamcs.mdb.model.Parameter property*), 20
- `unshelve_alarm()` (*yamcs.tmtc.client.ProcessorClient method*), 29
- `update_cop1_config()` (*yamcs.link.client.LinkClient method*), 54
- `update_type()` (*yamcs.tmtc.model.AlarmUpdate property*), 32
- `upload()` (*yamcs.filetransfer.model.Service method*), 62
- `upload()` (*yamcs.storage.model.ObjectInfo method*), 59
- `upload_object()` (*yamcs.storage.model.Bucket method*), 59
- `upload_object()` (*yamcs.storage.StorageClient method*), 58
- `UserInfo` (*class in yamcs.model*), 10
- `username` (*yamcs.core.auth.Credentials attribute*), 11
- `username()` (*yamcs.model.UserInfo property*), 10
- `username()` (*yamcs.tmtc.model.CommandHistory property*), 33
- `username()` (*yamcs.tmtc.model.IssuedCommand property*), 34
- `utc()` (*yamcs.tco.model.TCOCoefficients property*), 66
- `utc()` (*yamcs.tco.model.TCOSample property*), 66
- ## V
- `v_s()` (*yamcs.link.client.Cop1Subscription property*), 54
- `v_s()` (*yamcs.link.model.Cop1Status property*), 55
- `validity_duration()` (*yamcs.tmtc.model.ParameterValue property*), 36
- `validity_status()` (*yamcs.tmtc.model.ParameterValue property*), 36
- `value()` (*yamcs.archive.model.ColumnData property*), 47
- `value()` (*yamcs.mdb.model.EnumValue property*), 19
- `value_cache` (*yamcs.tmtc.client.ParameterSubscription attribute*), 31
- `vc_id()` (*yamcs.link.model.Cop1Config property*), 55
- `VerificationConfig` (*class in yamcs.tmtc.model*), 37
- `version()` (*yamcs.model.ServerInfo property*), 9
- `violation_count()` (*yamcs.tmtc.model.Alarm property*), 32
- ## W
- `WebSocketSubscriptionFuture` (*class in yamcs.core.futures*), 12
- `window_width()` (*yamcs.link.model.Cop1Config property*), 55
- ## Y
- `yamcs.archive.model` (*module*), 47
- `yamcs.core.auth` (*module*), 11
- `yamcs.core.exceptions` (*module*), 11
- `yamcs.core.futures` (*module*), 12
- `yamcs.filetransfer.model` (*module*), 61
- `yamcs.link.model` (*module*), 55
- `yamcs.mdb.model` (*module*), 17
- `yamcs.model` (*module*), 7
- `yamcs.storage.model` (*module*), 58
- `yamcs.tco.model` (*module*), 66
- `yamcs.tmtc.model` (*module*), 31
- `YamcsClient` (*class in yamcs.client*), 3
- `YamcsError`, 12