Yamcs Server Manual

Release 5.7.3-SNAPSHOT

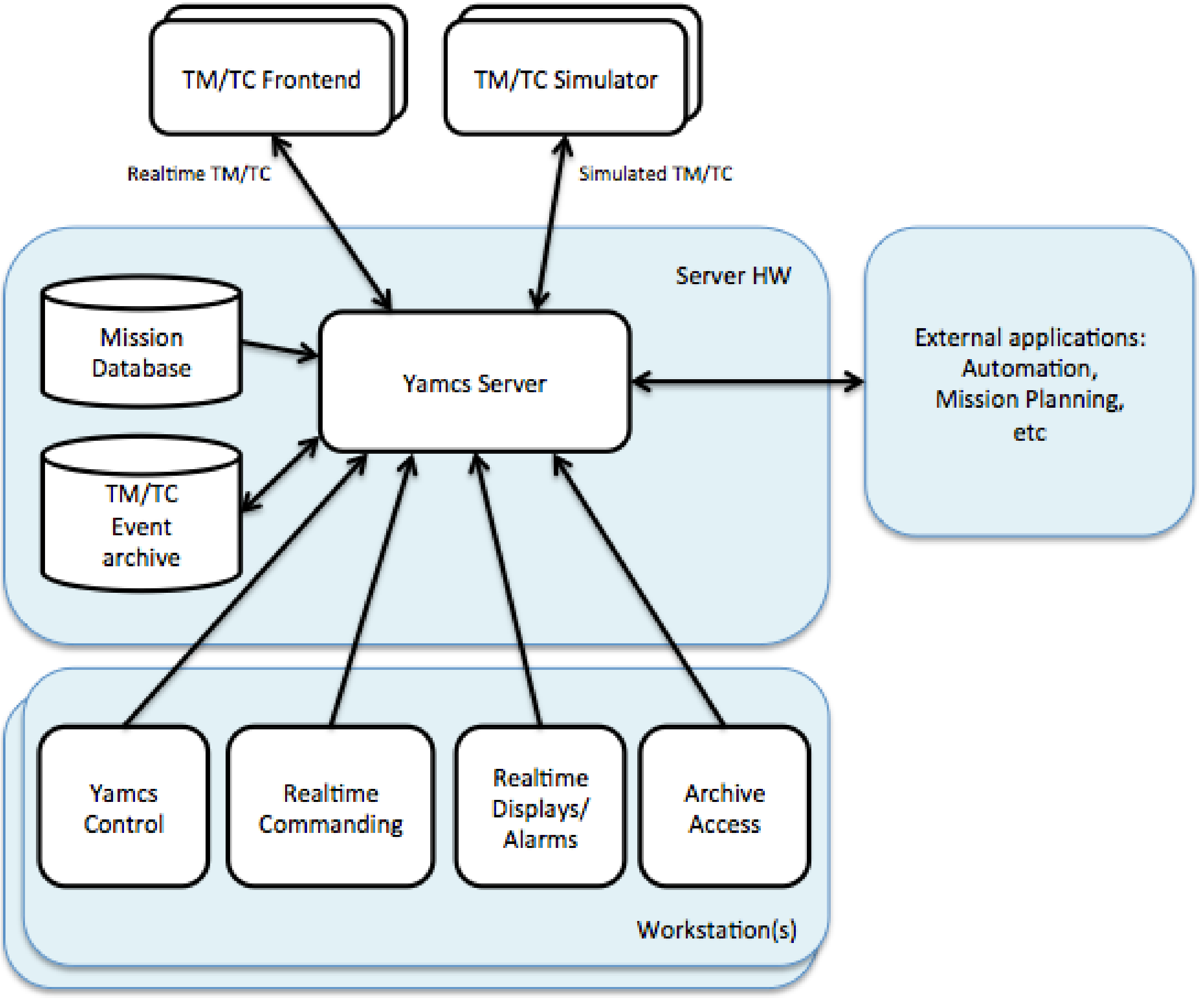
Space Applications Services, NV/SA
Leuvensesteenweg 325
1932 Sint-Stevens-Woluwe
Belgium
spaceapplications.com
yamcs.org

Aerospace Applications North America, Inc.
16850 Saturn Ln, Ste 100
Houston, TX 77058
United States of America
aerospaceapplications-na.com

Contents

1. General Information

Yamcs Server, or short Yamcs, is a central component for monitoring and controlling remote devices. Yamcs stores and processes packets, and provides an interface for end-user applications to subscribe to realtime or archived data. Typical use cases for such applications include telemetry displays and commanding tools.



Yamcs ships with an embedded web server for administering the server, the mission databases or for basic monitoring tasks. For more advanced requirements, Yamcs exposes its functionality over a well-documented HTTP-based API.

Yamcs is implemented entirely in Java, but it does rely on an external storage engine for actual data archiving. Currently the storage engine is RocksDB[1]. The preferred target platform is Linux x64, but Yamcs can also be made to run on Mac OS X and Windows.

[1] http://rocksdb.org/

1.1 Monitoring and Control Model

Yamcs implements a fairly traditional Monitoring and Control Model. The remote system is represented through a set of **parameters** which are sampled at regular intervals. Yamcs assumes that parameters are not sent individually but in groups which usually (but not necessarily) are some sort of binary packets. Yamcs supports basic parameter types (int, long, float, double, boolean, timestamp, string, binary) and also aggregate types (aka structs in C language) and arrays.

Parameters can either be received directly from the remote device or can be computed locally by **algorithms**. Algorithms in Yamcs can be implemented in Javascript or Python. Other languages that have JVM (Java Virtual Machine) based implementations could also be supported without too much trouble.

Following XTCE conventions, Yamcs distinguishes between **telemetered parameters** (= coming from remote devices), **derived** parameters (= computed by algorithms inside Yamcs), **local** parameters (= set by end-user applications) and **constant parameters** (which are just constant values defined in the mission database). In addition to these XTCE inspired parameter types, Yamcs defines **system parameters** (parameters generated by components inside Yamcs), **command** and **command history parameters**. The last two are specially scoped parameters that can be used in the context of command verifiers.

The parameters have limits associated to them and when those limits are exceeded, an **alarm** is triggered. The limits can change depending on the **context** which represent the state of the remote device. The context itself is derived from the value of other parameters.

An operator is informed of the triggered alarm in various ways depending on the end user application connected to Yamcs (e.g. red background in a display, audible alarm, sms, phone call, etc). After understanding the problem, the operator **acknowledges** the alarm, which means that it informs Yamcs that the alarm will be taken care of. This action - depending again on the remote end user application connected to Yamcs - means that other operators are not bothered anymore by the alarm. After the alarm has been acknowledged and the parameter goes back into limits, the alarm is **cleared** which means it is not triggered anymore. Before the alarm is acknowledged by an operator, it will stay triggered even if the parameter goes back into limits. An exception to this case is auto-acknowledging alarms which are cleared automatically when the parameter that triggered them goes back in limits.

As the parameters are supposed to be sampled regularly, they also have an expiration time. After the time is exceeded, the parameters become expired - that is to say at that time the state of the remote device is considered unknown.

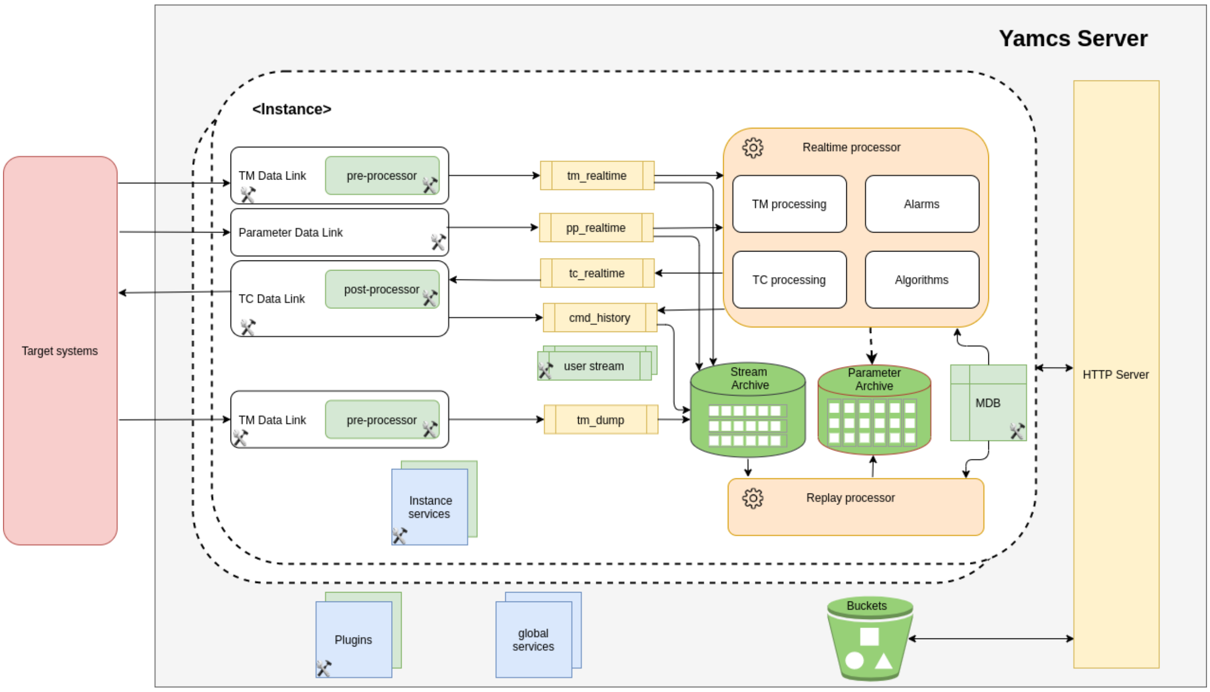
The remote device is controlled through the use of **(tele)-commands**. A telecommand is made up by a name and a number of **command arguments**. In order for a command to be allowed to be sent, the **command transmission constraints** (if any) have to be met. The constraints are expressed by the state of parameters (e.g. a command can be send only if a subsystem is switched on). Some commands can have an elevated **significance**, which may mean that a special privilege or an extra confirmation is required to send the command. Once the command has been sent, it passes through a series of execution stages. XTCE pre-defines a series of stages (TransferredToRange, SentFromRange, Received, Accepted, etc). Yamcs does not enforce the use of these predefined stages, the user is free to choose any number of random stages. Each stage has associated a **command verifier** - this is an algorithm that will decide if the command has passed or not that stage. It is also possible to specify that the stage has been passed when a specific packet has been received.

The command text (command name and argument values), the binary packet (if binary formatted) and the different stages of the execution of the command are recorded in the **command history**. Yamcs does not limit the information that can be added to the command history. This can be extended with and arbitrary number of (key, timestamp, value) attributes.

1.2 Server Architecture

The Yamcs server runs as a single Java process and it incorporates an embedded HTTP server implemented using Netty.

The main components are depicted in the diagram below.



1.2.1 Instances

The Yamcs instances provide means for one Yamcs server to monitor/control different payloads or satellites or version of the payloads or satellites at the same time.

Most of the components of Yamcs are instance-specific.

1.2.2 Data Links

Data Links are components that connect to the target system (instruments, ground stations, lab equipment, etc). One Yamcs instance will contain multiple data links. There are three types of data received/sent via Data Links:

- Telemetry packets. These are usually binary chunks of data which have to be split into parameters according to the definition into a Mission Database.
- Telecommands. These are usually the reverse of the telemetry packets.
- Parameters. These are historically (ISS ground segment) called also processed parameters to indicate they are processed (e.g. calibrated, checked against limits) by another center.

Connecting via a protocol to a target system means implementing a specific data link for that protocol. In Yamcs there are some built-in Data Links for UDP and TCP. SLE (Space Link Extension) data links are also implemented in a plugin.

The pre-processors run inside the TM data links and are responsible for doing basic packet processing (e.g. verifying a CRC or checksum) which is not described in the Mission Database.

The post-processor runs inside the TC data link and are responsible for doing command processing (e.g. computing a CRC or checksum) which is not described in the Mission Database.

Please note in the picture above that while for telecommands there is a link sending realtime data, for telemetry we also have a data link retrieving dump data - this is data that has been recorded somewhere (on the spacecraft or some other intermediate point) and dumped later. Usually there is no continous visibility of the spacecraft from the ground and thus most spacecrafts are capable of recording data onboard. The dump data will not be sent to the relatime displays (because the display shows the realtime data coming in paralell) but it will be sent to the archive where it has to be merged with the old data and with the realtime incoming data.

Yamcs does not define the dump data as a special type of data, it is the configuration of which data is sent on which stream and which stream is connected to which processor (see below what streams and processors are) that determines what dump data is.

The CCSDS standards specify a higher level entity called transport frame. Typically the telemetry transfer frames are fixed size and the variable size packets are encoded in the fixed size frames using a well defined procedure. The packets can be multiplexed on the same transmission channel with other types of data such as a video bitstream. The frames allows also multiplexing realtime data with dump data. In order to maintain a constant bitrate required for RF communication, the standards also define the idle data to be sent when no other data is available.

In Yamcs, all the CCSDS frame processing is performed at the level of Data Links - when frame procesisng is used, there is a data link that receives the frame (e.g. via SLE) and then demultiplexes it into multiple sub-links which in turn apply the pre-processor for TM and send the data on the streams to the processors and archive. There is a sub-link (or more) for realtime data and similarly a sub-link (or more) for dump data. Yamcs handles packets and parameters, other type of data (e.g. video) could be sent to external systems for processing and storage.

1.2.3 Streams

Streams are components inside Yamcs that transport tuples. They are used to de-couple the producers from the consumers, for example the Data Links from the Processors. The de-coupling allows the user to change the data while being passed from one component to another.

1.2.4 Processors

The Yamcs processor is where most of the monitoring and control functions takes place: packets get transformed into parameters, limits are monitored, alarms are generated, commands are generated and verified, etc. There can be multiple processors in one instance, typically one permanently processing realtime data and other created on demand for replays.

In particular, the Parameter Archive will create regularly a processor for parameter archive consolidation. This is required in order to process the data received in dump mode (see above) which does not pass through a realtime processor.

1.2.5 Mission Database (MDB)

The Mission Database contains the description of the telecommands and telemetry including calibration curves, algorithms, limits, alarms, constraints, command pre and post verification.

1.2.6 Services

A service in Yamcs is a Java class that implements the org.yamcs.YamcsService[2] interface. The services can be:

- global meaning they run only once at the level of the server; their definition can be found in `yamcs.yaml`. One such service is the HttpServer.
- instance specific meaning that they run once for each Yamcs instance where they are included; their definition can be found in `yamcs.<instance>.yaml`

[2] https://yamcs.org/javadoc/yamcs/org/yamcs/YamcsService.html

- processor specific meaning they run at the level of the processor; their definition can be found in `processor.yaml`.

User can define their own services by adding a jar with an implemented java class into the Yamcs lib/ext directory.

1.2.7 Plugins

A plugin in Yamcs is a Java class that implements the org.yamcs.Plugin[3] interface. The plugin classes are loaded by the Yamcs server at startup before starting any instance. Although not required, it is advised that the user creates a plugin with each jar containing mission specific functionality. This will allow to see in the Yamcs web the version of the plugin loaded; the plugin is also the place where the user can register new API endpoints.

1.2.8 Stream Archive

The Stream Archive is where tuples can be stored. This is a realtime archive, data is inserted as soon as it is received from a stream. It is optimized for storing data sorted by time.

1.2.9 Parameter Archive

The Parameter Archive contains values of parameters and is optimized for retrieving the value for a limited set of parameters over longer time intervals. The archive is not realtime but is obtained by creating regular replays transforming data from the stream archive via a processor. Whereas the basic storage unit of the stream archive corresponds to data at one specific time instant (e.g. a telemetry packet, a set of parameters with the same timestamp), the basic storage unit of the parameter archive is a set of values of one parameter over a time interval.

1.2.10 Buckets

Buckets are used for storing general data objects. For example the CFDP service will store there all the files received from the on-board system. As for most Yamcs components, there is an REST API allowing the user to work with buckets (get, upload, delete objects).

1.2.11 Extension points

In the diagram above, there are some components that have a build symbol; these is where we expect mission specific functionality to be added:

- new data links have to be implemented if the connection to the target system uses a protocol that is not implemented in Yamcs.
- packet pre-processor and command post-processor are componenets where the user can implement some specific TM/TC headers, time formats etc.
- the Mission Database (MDB) contains the description of telecommands and telemetry and is entirely mission specific.
- user defined streams can implement command routing or basic operations on packets (e.g. extracting CLCW from a TM packet).
- user defined services can add complete new functionality; an example of such functionality is to assemble telemetry packets into files (this is what the CFDP service does, but if the user's system does not use CFDP, a new service can be developed).

[3] https://yamcs.org/javadoc/yamcs/org/yamcs/Plugin.html

- finally plugins can be used to group together all the mission specific functionality.

1.3 Time in Yamcs

The text below documents several aspects of working with time in Yamcs.

1.3.1 Time Encoding

Yamcs uses 8 bytes signed integers (long in java) for representing milliseconds since 1-Jan-1970 00:00:00 TAI, including leap seconds. The Yamcs time in milliseconds is the UNIX time (in milliseconds) + leap seconds.

In practice, in 2022, the following is true:

```
TimeEncoding.getWallclockTime() = System.currentTimeMillis() + 37000.
```

The leap seconds are loaded from the file UTC-TAI.history in the Yamcs etc directory. This file is part of the so called IERS (International Earth Rotation and Reference Systems Service) Bulletin C and is published here: https://hpiers.obspm.fr/iers/bul/bulc/UTC-TAI.history.

The user is responsible for updating manually this file if it changes (when new leap seconds are added). Fortunately this is not very often and new leap seconds are announced well in advance; for instance there has been no new leap second between 2017 and 2022.

Yamcs also has a high resolution time implemented in the class org.yamcs.time.Instant[4]. This is represented as 8+4 bytes milliseconds and picoseconds of the millisecond. It is not widely used - in Java it is not even easily possible to get the time with a resolution better than millisecond.

The higher resolution time is sent sometimes from external systems - for example a Ground Station may timestamp the incoming packets with a microsecond or nanosecond precise time (derived from an atomic clock) - this time is available as the Earth Reception Time via the yamcs-sle plugin.

1.3.2 Mission Time

The mission time in Yamcs is the “current” time - for a realtime mission that would be the wall clock time (in practice the computer time); for a simulation it would be the simulation time.

The mission time is specific to a Yamcs instance and is given by the org.yamcs.time.TimeService[5] configured in that instance. The time service is configured using the `timeService` keyword in the yamcs.instance.yaml.

There are two time services implemented as part of standard Yamcs:

- org.yamcs.time.RealtimeTimeService[6] - it uses always the wall clock time (the computer time) as the mission time.
- org.yamcs.time.SimulationTimeService[7] - this allows to run a simulated time at arbitrary speeds. The time can be set externally via the Time API[8] or from a TM data link. Since Yamcs 5.6.1 it is possible to synchronize the mission time between two instances on two different Yamcs servers via the replication service.

Plugins may come with their own implementation of a time service. For example the yamcs-eurosim plugin gets the mission time from Eurosim (a simulation framework).

[4] https://yamcs.org/javadoc/yamcs/org/yamcs/time/Instant.html
[5] https://yamcs.org/javadoc/yamcs/org/yamcs/time/TimeService.html
[6] https://yamcs.org/javadoc/yamcs/org/yamcs/time/RealtimeTimeService.html
[7] https://yamcs.org/javadoc/yamcs/org/yamcs/time/SimulationTimeService.html
[8] https://docs.yamcs.org/yamcs-http-api/time/set-time

1.3.3 Processor Time

The processor time is the time visible in the yamcs-web. For realtime processors is the same as the mission time; for replay processors is the time of the replay - it is extracted from the packets or parameters as they are extracted from the archive.

1.3.4 Reception Time

The reception time is the time associated to data (packets, parameters, events) as it comes into Yamcs. The reception time is always set to mission time.

1.3.5 Generation Time

The generation time is the time when the data has been generated.

For telemetry packets, it is set by the pre-processor, normally with a time extracted from the packet. However it can be set to the mission time if the `useLocalGenerationTime` option is set to true.

The timeEncoding option is used on the TM links to configure how to extract the time from the packet - which means how to covert a number (or more numbers) extracted from the packet to a Yamcs time. The various options for time decoding are documented in the *Packet Pre-processor* (page 56)

The spacecrafts which have no mean to syncrhonize time (e.g. no access to GPS) will usually use a free running on-board clock (initialized to 0 at startup) to timestamp the packets. In these cases, the on-board time needs to be correlated with the mission time. The *Time Correlation Service* (page 104) can be used for this purpose.

Finally, the TM links have an option `updateSimulationTime` which can be used to set the mission time to the time extracted from the packet. This works if the SimulationTimeService is used.

1.3.6 Earth Reception Time

The earth reception time is the time a TM packet has been received in a ground station. The TM links are responsible for setting this on the packet inside Yamcs. For example the SLE TM link (part of the yamcs-sle plugin) will receive the earth reception time via the SLE protocol.

The earth reception time is a high resolution time which may be used in the process of time correlation.

2. Server Administration

2.1 Configuration

Yamcs configuration files are written in YAML format. This format allows to encode in a human friendly way the most common data types: numbers, strings, lists and maps. For detailed syntax rules, please see https://yaml.org.

The root configuration file is `etc/yamcs.yaml`. It contains a list of Yamcs instances. For each instance, a file called `etc/yamcs.instance-name.yaml` defines all the components that are part of the instance. Depending on which components are selected, different configuration files are needed.

2.1.1 Server Configuration

The number of configuration options in `etc/yamcs.yaml` are relatively limited. A sample configuration file is below.

```yaml
services:
    - class: org.yamcs.http.HttpServer

instances:
    - simulator

dataDir: /storage/yamcs-data

incomingDir: /storage/yamcs-incoming


secretKey: "changeme"

yamcs-web:
    tag: DEMO
```

The following options are supported

services (list)
A list of global services. Users can create their own global services that are unique for the whole Yamcs instance. The global services description can be found in *Global Services* (page 89)

instances (list)
A list of instances loaded at Yamcs start. It is also possible to load instances from <dataDir>/instance-def directory. The instances created created via the API will be stored there.

dataDir (string)
A directory which will be the root of the Yamcs archive. The directory must exist and it shall be possible for the user who runs Yamcs to write into it. More information about the Yamcs archive can be found in *Data Management* (page 45). In addition to the directories used for the archive, there are two directories named `instance-def` and `instance-templates` which are used for the dynamic creation of instances.

incomingDir (string)

A directory used by the FilePollingTmDataLink[9] to load incoming telemetry files. This is a relic from when Yamcs was used only for that; the option should be specified in the link configuration instead (it is left here because some users are quite accustomed to it).

secretKey (string)

A key that is used to sign the authentication tokens given to the users. It should be changed immediately after installation. As of version 5.0.0, Yamcs does not support persisted authentication tokens but this feature will be available in a future version.

yamcs-web (map)

Configuration of the yamcs web application. The different options are documented in *Web Interface* (page 125)

2.1.2 Instance Configuration

The instance configuration file `yamcs.<instance-name>.yaml` contains most of the options that need to be set on a Yamcs server.

```yaml
services:
    - class: org.yamcs.archive.XtceTmRecorder
    ...

dataLinks:
    - name: tm_realtime
      enabledAtStartup: false
      class: org.yamcs.tctm.TcpTmDataLink
      ...

mdb:
    - type: "sheet"
      spec: "mdb/simulator-ccsds.xls"
      subloaders:
          - type: "sheet"
            spec: "mdb/simulator-tmtc.xls"
      ...

streamConfig:
    tm:
      - name: "tm_realtime"
        processor: "realtime"
      - name: "tm2_realtime"
        rootContainer: "/YSS/SIMULATOR/tm2_container"
        processor: "realtime"
      - name: "tm_dump"
    cmdHist: ["cmdhist_realtime", "cmdhist_dump"]

timeService:
    class: org.yamcs.time.SimulationTimeService
```

The following options are supported

services (list)

A list of instance specific services. Each service is specified by a class name and arguments which are passed to the service at initialization. Services are implementations of YamcsService[10]. Users can create their own services; most of the missions where Yamcs has been used required the creation of at least a mission specific service. More description of available services can be found in *Instance Services* (page 96).

dataLinks (list)

A list of data links - these are components of Yamcs responsible for receiving/sending data to a target system. Sometimes users need to create additional data links for connecting via different protocols (e.g. MQTT). The available data links are documented in *Data Links* (page 55)

[9] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/FilePollingTmDataLink.html

[10] https://yamcs.org/javadoc/yamcs/org/yamcs/YamcsService.html

mdb (list)

The configuration of the Mission Database (MDB). The configuration is hierarchical, each loader having the possibility to load sub-loaders which become child Space Systems. More information about the MDB can be found in *Mission Database* (page 13)

streamConfig(map)

This configures the list of streams created when Yamcs starts. The map contains an entry for each standard stream type (`tm`, `cmdHist`, `event`, etc) and additionally a key `sqlFile` can be used to load a StreamSQL file where user defined streams can be created. More information can be found in *Streams* (page 45)

timeService(map)

This configures the source of the “mission time”. By default the RealtimeTimeService uses the local comuter clock as the time source. The *javadoc:org.yamcs.time.SimulationTimeService* can be used to simulate a mission time in the past or the future. If configured, the time can be controlled using the REST API. The `updateSimulationTime: true` option on a telemetry data link can also be used to manipulate the simulation time - in this case the time will be set to be the generation time of the packet.

2.2 Logging

Yamcs allows capturing runtime log messages at different verbosity levels to different output handlers.

By default, if unconfigured, Yamcs will emit messages at INFO level to stdout.

The `yamcsd` program acceps some options to modify these defaults. In particular:

`--log`

The numeric verbosity level, where 0 = OFF, 1 = WARNING, 2 = INFO, 3 = FINE and 4 = ALL. Default: 2

`--log-config`

Detailed logger verbosity levels. If unspecified, the `--log` option impacts all loggers, which may lead to excessive output.

`--no-color`

Turn off ANSI color codes

If the configuration directory of Yamcs includes a file `logging.properties`, then logging properties are read from this file instead of applying the default console logging. Logging-related program arguments (e.g. verbosity) are then ignored.

The `logging.properties` uses the standard Java logging format, which allows to tweak the logging in much more detail than what is possible through the command-line flags of the yamcsd executable.

A full description of the syntax is beyond the scope of this manual, but see this example of how we currently configure our generic RPM packages:

logging.properties

```
handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler

java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = org.yamcs.logging.JournalFormatter
java.util.logging.ConsoleHandler.filter = org.yamcs.logging.GlobalFilter

java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.pattern = /opt/yamcs/log/yamcs-server.log
java.util.logging.FileHandler.limit = 20000000
java.util.logging.FileHandler.count = 50
java.util.logging.FileHandler.formatter = org.yamcs.logging.CompactFormatter

org.yamcs.level = FINE
```

There are two handlers:

1. A ConsoleHandler prints its messages to stdout. The console output can for example be consumed by an init system like systemd. This configuration uses a JournalFormatter that prints short messages without timestamp for direct injection into the systemd journal, it also applies a GlobalFilter that will remove log messages specific to an instance. This makes Yamcs less chatty.
2. A FileHandler defines the properties used for logging to `/opt/yamcs/log/yamcs-server.log.x`. The FileHandler in this configuration applies a rotation 20MB with a maximum of 50 files. The theoretic maximum of disk space is therefore 1GB. The most recent log file can be found at `/opt/yamcs/log/yamcs-server.log.0`. Note that when Yamcs Server is restarted the log files will always rotate even if `yamcs-server.log.0` had not yet reached 20MB.

This configuration logs messages coming from `org.yamcs` loggers at maximum FINE level. Each handler may apply a further level restriction. This is applied after the former level restriction. For example the above FileHandler has level ALL, however it will never print messages more verbose than FINE.

For specific use cases, Yamcs includes a few custom loggers:

`org.yamcs.logging.SyslogHandler`

Writes to syslogd over UDP with messages formatted according to RFC 3164 (BSD syslog).

The formatting of this handler cannot be modified, and does not include full stacktrace information.

`org.yamcs.logging.SyslogHandler.level`
Minimum level of loggable messages. Default: ALL.

`org.yamcs.logging.SyslogHandler.host`
Syslog host. Defaults to loopback.

`org.yamcs.logging.SyslogHandler.port`
Syslog port. Default: 514.

`org.yamcs.logging.SyslogHandler.facility`
Syslog facility. Default: 1 (= user-level messages)

`org.yamcs.logging.WatchedFileHandler`

Handler that watches the file that it is logging to. When that file is deleted, the handler will close and reopen a new file with the same name. This handler is designed to be used with programs like `logrotate` that take care of log rotation outside of the JVM. Without the watch functionality, log messages would just continue to be written to the old (rotated) file.

`org.yamcs.logging.WatchedFileHandler.level`
Minimum level of loggable messages. Default: ALL.

`org.yamcs.logging.WatchedFileHandler.filename`
Name of the file that is logged to. Default: yamcs.log

`org.yamcs.logging.WatchedFileHandler.filter`
Instance of `java.util.logging.Filter`. Default: unfiltered

`org.yamcs.logging.WatchedFileHandler.formatter`
Instance of `java.util.logging.Formatter`. Default: `java.util.logging.XMLFormatter`

3. Mission Database

The Mission Database describes the telemetry and commands that are processed by Yamcs. It tells Yamcs how to decode packets or how to encode telecommands.

The database organizes TM/TC definitions by **space system**. A space system may contain other sub-space systems, thereby structuring the definitions in logical groups. Space systems have a name and can be uniquely identified via UNIX-like paths starting from the root of the space system hierarchy. For example: `/BogusSAT/SC001/BusElectronics` could be the name of a sub-space system under `/BogusSAT/SC001`. The root space system is `/`.

The terminology used in the Yamcs Mission Database is very close to the terminology used in the XTCE exchange format. XTCE prescribes a useful set of building blocks: space systems, containers, parameters, commands, algorithms, etc.

3.1 Data Types

The MDB data types are associated to parameters and command arguments and provide several characteristics of these:

- the value type (int64, int32, float,. . .) of the engineering value
- the value type of the raw value
- validity conditions
- units
- alarms (only for types corresponding to parameters)
- engineering/raw transformation using calibrators
- raw/binary transformation using data encodings

The distinction between a parameter and its type is not so evident and many control systems do not make this distinction (i.e. each parameter with its own type).

In practice most use of shared types has been to define generic types such as “uint8”, “uint16”, etc and use those for parameters that do not require any calibration, units or other specific properties. Types can also be shared for parameters associated to the same type of sensors which do not need individual calibrators.

Yamcs supports the following parameter and argument data types:

- Integer data type
- Float data type
- Boolean data type
- String data type
- Binary data type

- Absolute time data type
- Enumerated data type - a (integer, string) pair.
- Aggregate data type - complex data type similar to a C-struct. Each member of the aggregate has a name and a type.
- Array data type- multidimensional array where each element is of the same type.

As mentioned above, one important function of a data type is to describe how to represent the raw value on the wire (i.e. in the command or telemetry packet). The following encodings are supported:

- Integer data encoding
- Float data encoding
- Boolean data encoding
- String data encoding
- Binary data encoding

Note that `xyz` in the `xyz data encoding` refers to the type of the raw value whereas the `xyz` in `xyz data type` refers to type of the engineering value.

One will certanly notice that there is no direct encoding for absolute times, enumerated, aggregated and array value types. Currently these can only be encoded/decoded by other means (e.g. an aggregate value will be decoded by decoding its members, an enumerated value by decoding its integer or string representation).

The integer and float encodings have optionally a calibrator which allow transforming the raw value to engineering value or reverse.

There may be MDB data types without encoding - these are used by local parameters which are never encoded on wire.

All the data encodings in Yamcs can be performed by user defined java code by implementing the org.yamcs.mdb.DataEncoder[11] or org.yamcs.mdb.DataDecoder[12] respectively. Such code has to be written if the encoding format is not part of Yamcs.

3.1.1 Parameter types vs Argument types

The data types described in this section are used both for parameters and command arguments. Internally in Yamcs the types are not shared.

For convenience, when defining the Mission Database in spreadsheet format, there is one place where all the data types are defined. However when Yamcs loads the spreadsheet, it duplicates in memory the definition for the parameters and arguments.

In XTCE they are defined in different sections: `<ParameterTypeSet>` and `<ArgumentTypeSet>`.

Note that the calibrator (if defined) applies in a different direction: for parameter types it converts from raw to engineering value whereas for argument types it converts from engineering value to raw. Thus one cannot apply the same calibrator even if a parameter corresponds conceptually to an argument. The user would have to invert (in mathematical terms) the calibrator used in the parameter type definition when defining the corresponding argument data type.

3.1.2 Integer data type

Integer values in Yamcs can be 32 or 64 bits signed or unsigned.

[11] https://yamcs.org/javadoc/yamcs/org/yamcs/mdb/DataEncoder.html
[12] https://yamcs.org/javadoc/yamcs/org/yamcs/mdb/DataDecoder.html

Integer values can be encoded/decoded on any number of bits smaller than 64. Signed and unsigned values are supported. Signed values can be encoded in twos complement, sign magnitude or ones complement.

A simple XTCE example of an unsigned integer parameter type with an integer encoding:

```
<IntegerParameterType signed="false" name="uint16">
    <IntegerDataEncoding encoding="unsigned" sizeInBits="16" />
    <ValidRange minInclusive="100" maxInclusive="1000"/>
</IntegerParameterType>
```

Note that by default the type has a `sizeInBits=32` so the value will be converted from 16 bits on the wire to 32 bits value. Yamcs will use a 32 bit integer for any parameter with `sizeInBits <= 32` and a 64 bit integer for any type with the `32 < sizeInBits <= 64`.

The `<ValidRange>` construct is optional and used differently for parameters and arguments:

- for parameters it is used to check the validity. If a parameter value does not satisfy the range, it will be marked as invalid (and can be seen with a specific color in the display)
- for arguments it is used to verify the value provided by the user. If the value does not match the range, the command is rejected.

Integer parameters can also have associated alarms and calibrators (see below an example for float parameters, it is identical for integer parameters).

One important thing to mention about calibrators is that even when associated to the integers, they still work on (signed) double floating point numbers. Some precision will be lost when converting from a large (unsgined) integer to a double or viceversa.

The integer parameters can also be encoded as strings, as in the following XTCE example:

```
<IntegerParameterType signed="false" name="int_encoded_as_string">
     <StringDataEncoding>
         <SizeInBits>
             <Fixed>
                 <FixedValue>48</FixedValue>
             </Fixed>
             <TerminationChar>00</TerminationChar>
         </SizeInBits>
     </StringDataEncoding>
</IntegerParameterType>
```

In this case the raw value will be of type string and the engineering value of type integer. For an explanation of how the string encoding works, please see below in the String data type section.

3.1.3 Float data type

Floating point data in Yamcs can be simple precision (32 bit) or double precision (64 bit).

It can be encoded/decoded either to a IEEE754 representation or to an integer representation using a calibration function. Typically a sensor will produce a digital value (e.g. 12 bits integer) which has to be converted to an analog value using a calibration (or transfer) function.

An XTCE example of a float parameter encoded as integer and having a polynomial calibrator:

```
<FloatParameterType initialValue="20" name="Temperature_Type">
    <UnitSet>
        <Unit>degC</Unit>
    </UnitSet>
    <IntegerDataEncoding encoding="unsigned" sizeInBits="12">
        <DefaultCalibrator>
            <PolynomialCalibrator>
                <Term coefficient="0" exponent="-20" />
                <Term coefficient="1" exponent=".025" />
            </PolynomialCalibrator>
        </DefaultCalibrator>
```

(continues on next page)

(continued from previous page)

```
    </IntegerDataEncoding>
    <DefaultAlarm>
        <StaticAlarmRanges>
            <WarningRange minInclusive="10" maxInclusive="30" />
            <CriticalRange minInclusive="-10" maxInclusive="50" />
            </StaticAlarmRanges>
    </DefaultAlarm>
</FloatParameterType>
```

Yamcs supports the following type of calibrations:

- polynomial - the conversion between the raw value and the engineering value is obtained by applying a polynomial function.
- linear spline (point pairs) - the conversion between the raw and engineering value is obtained by interpolating linearly the raw value.
- mathematical operations specified in reverse polish notation (only in XTCE format) - the conversion is obtained by applying the mathematical operation.
- Java expressions (only in spreadsheet format) - the conversion is obtained by running it through the java expression.

The java expression is the most flexible calibration as it can practically call any java code available on the server. However it is not allowed by XTCE (instead an algorithm can be used to generate the output value into a different parameter).

The example above also defines an default alarm - perhaps a bit counter intuitive the parameter will trigger the alarm if it is outside of the range defined there (for example a value of 40 will trigger the warning alarm and a value of -15 will trigger the critical alarm). As per XTCE there are 5 levels of alarms supported (in order of severity): watch, warning, distress, critical and severe.

Both calibrators and alarms can be contextualized: that means a different alarm or calibrator will be used depending on the value of other parameters.

While the most common encoding for float is float encoding, the other encodings can also be used:

- integer: will convert number to integer by performing a java cast to long and then fitting the long into the number of bits required. This may result in loss of precision and even in completely wrong number when converting a signed float to a unsigned integer.
- string: the value will be converted to a string representation.
- binary:

3.1.4 Boolean data type

Boolean values in Yamcs take take a simple `true` or `false` value. In XTCE one can define different values instead of `true`/`false` as in the example below. Yamcs only supports these values when reading the XTCE file (they can be used in conditions for example) but the value computed does not include the string (and thus cannot be shown in the display).

To encode boolean values one can use any data encoding with the following transformations:

- **for integer/float raw values:**
  - decoding: `0` is `false` and anything else is true when decoding.
  - encoding: `true` is converted to `1`, `false` is converted to `0`.

- **for string values:**
  - decoding: if the string value is empty, case insensitive equal with the `zeroStringValue` defined in the type or with the string `0` then the value is `false`, anything else is `true`.
  - encoding: `true` is converted to the `oneStringValue` defined in the type, `false` is converted to `zeroStringValue` defined in the type.
- **for binary values:**
  - decoding: if the binary value is empty or consists only of nulls then the value of the boolean is `false` anything else is `true`.
  - encoding: the value is converted to a binary array of one element with the value `1` if `true` or `0` if `false`.

```
<BooleanParameterType name="bool2" oneStringValue="yes!" zeroStringValue="nooo">
    <StringDataEncoding>
        <SizeInBits>
            <Fixed>
                <FixedValue>32</FixedValue>
            </Fixed>
            <TerminationChar>00</TerminationChar>
        </SizeInBits>
    </StringDataEncoding>
</BooleanParameterType>
```

The spreadsheet format allows to define a data type with a boolean data encoding by using a raw type of `bool` in the Data Type definition. This encoding is not possible to be defined in XTCE (but it is equivalent with a 1 bit integer encoding) and it always uses one bit representation with `0 = false` and `1 = true`.

3.1.5 String data type

In Yamcs the string data is represented as a java (unicode) String value. The encoding to/from the wire is performed using a string data encoding with one of the supported Java Charsets[13] (UTF-8, ISO-8859-1, etc)

In addition to converting the bytes to unicode character, a typical problem in decoding telemetry is to know the boundary of the string inside the packet. To comply with XTCE Yamcs implements a “string in a buffer” approach:

- conceptually the packet contains a buffer (or a box) where the string has to be extracted from or encoded into.
- the buffer can be the same size with the string or larger than the string. If the buffer is larger than the string, it will be filled by Yamcs with 0 for commands or some filler which is ignored by Yamcs for telemetry.
- if the buffer is larger than the string, the buffer size can be fixed or its size can be determined from the value of a parameter/argument.
- inside the buffer:
  - the string can fill completely the buffer (so the size of the string is determined by the size of the buffer).
  - the size of the string can be encoded at the beginning of the buffer (in front of the string)
  - or the string can be terminated by a special character (or by the end of the buffer, whichever comes first).

One case which is not supported by Yamcs (nor by XTCE) is a fixed size string inside a fixed size buffer with the string not filling completely the buffer. For this case you can limit the size of the buffer to the size of the

[13] https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html

string and define another parameter for the remaining of the buffer, or simply define an offset for the next container entry.

The size of the buffer is in number of bytes - depending on the encoding used, a character of the string may be encoded on multiple bytes (for example UTF-8 encodes each character in one to four bytes).

Finally, please note that although XTCE defines a number of bits for the buffer size or for the size tag, Yamcs only supports encoding these on an integer number of bytes (e.g. encoding strings on partial bytes is not supported) so the number of bits has to be divisible by 8.

Example 1: string encoded in a fixed size buffer with a null terminator

The buffer is 6 bytes long (meaning that the next parameter will come after the 6 bytes even if the string is shorter). If the terminator is not found, it is not considered an error and the string will be 6 bytes long. If the terminator is not specified (by removing the `<TerminationChar>` section), the string will always be 6 bytes long. Note that it may cause the string to include nulls but that is not a problem in Java.

```
<StringParameterType name="string1">
    <StringDataEncoding encoding="UTF-8">
        <SizeInBits>
            <Fixed>
                <FixedValue>48</FixedValue>
            </Fixed>
            <TerminationChar>00</TerminationChar>
        </SizeInBits>
    </StringDataEncoding>
</StringParameterType>
```

This example can be defined in the spreadsheet with the encoding `terminated(0x00, UTF-8, 48)`. If there is no terminator (so the string covers all the time the buffer), the equivalent spreadsheet encoding is `fixed(48, UTF-8)`.

Example 2: prefixed size string encoded in undefined buffer

The buffer is not explicitely defined so it is effectively as long as the prefix + string. The `maxSizeInBits` refers to the size of the buffer, so in this example the maximum size of the string will be 4.

Note the `_yamcs_ignore` parameter reference which is used to workaround XTCE mandating a dynamic value. Yamcs will accept the XML file without the `DynamicValue` section but the file will not validate with XTCE 1.2 xsd. An alternative for the `_yamcs_ignore` would be to derive the buffer length from the packet length.

```
<StringParameterType name="string5">
    <StringDataEncoding encoding="UTF-8">
        <Variable maxSizeInBits="48">
            <DynamicValue>
                <ParameterInstanceRef parameterRef="_yamcs_ignore" />
            </DynamicValue>
            <LeadingSize sizeInBitsOfSizeTag="16" />
        </Variable>
    </StringDataEncoding>
</StringParameterType>
```

This example can be best defined in the spreadsheet with the encoding `PrependedSize(16)`. The maximum size cannot be defined, so the effective maximum size will be the remaining of the packet.

Example 3: null terminated string encoded in undefined buffer

This examples provdides string argument type whose size is variable. The buffer is not defined which means the buffer will be effectively the string + terminator.

The maxSizeInBits refers to the maximum size of the buffer; it means that the maximum size of the string in binary is `maxSizeInBits/8 - 1`.

Note the `yamcs_ignore` parameter reference which is used to workaround XTCE mandating a dynamic value. Yamcs will accept the XML file without the `DynamicValue` section but the file will not validate with XTCE 1.2 xsd. An alternative for the `_yamcs_ignore` would be to define an argument for the buffer length but that would be inconvenient for the user.

```
<StringArgumentType name="string3">
    <StringDataEncoding encoding="UTF-8">
        <Variable maxSizeInBits="48">
            <DynamicValue>
                <ParameterInstanceRef parameterRef="_yamcs_ignore" />
            </DynamicValue>
            <TerminationChar>00</TerminationChar>
        </Variable>
    </StringDataEncoding>
</StringArgumentType>
```

More XTCE examples:

- yamcs-core/src/test/resources/xtce/strings-tm.xml[14]
- yamcs-core/src/test/resources/xtce/strings-cmd.xml[15]

More Spreadsheet examples:

- yamcs-core/mdb/refmdb.xls[16]

Finally, we mention that string values can also be encoded with a binary encoder; the translation from string to binary is using the String#getBytes[17] method.

3.1.6 Binary data type

A binary data type represents a sequence of bytes (a byte[] in java). The values of this type implicitly have a length.

As for strings, Yamcs only supports types which are an integer number of bytes.

Unlike strings, when encoding binary values there is no distinction between the value being encoded and the buffer in which the value is encoded: the value always fills the buffer.

Example 1: binary parameter type of fixed size

```
<BinaryParameterType name="binary_type1">
    <BinaryDataEncoding>
        <SizeInBits>
            <FixedValue>128</FixedValue>
        </SizeInBits>
    </BinaryDataEncoding>
</BinaryParameterType>
```

A parameter of this type will always be 16 bytes in length.

Example 2: binary parameter type of variable size with the size given by another parameter

The example below defines a parameter type whose size is given by another parameter named `size`. That parameter has to be of integer type and preceede the binary one in the packet.

[14] https://github.com/yamcs/yamcs/blob/master/yamcs-core/src/test/resources/xtce/strings-tm.xml
[15] https://github.com/yamcs/yamcs/blob/master/yamcs-core/src/test/resources/xtce/strings-cmd.xml
[16] https://github.com/yamcs/yamcs/blob/master/yamcs-core/mdb/refmdb.xls
[17] https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#getBytes

```
<BinaryParameterType name="BinaryType">
    <BinaryDataEncoding>
        <SizeInBits>
            <DynamicValue>
                <ParameterInstanceRef parameterRef="size" />
                <LinearAdjustment slope="8" />
            </DynamicValue>
        </SizeInBits>
</BinaryDataEncoding>
```

Note the `<LinearAdjustment>` construct which allows to convert from number of bytes to number of bits required by the `<SizeInBits>` element.

Example 3: binary argument type of variable size with the size encoded in front of the data

The example above needs another parameter for the data size. When used in command it has the disatvantage that the user needs to enter the number of bytes in addition to the bytes themselves (with the risk of introducing inconsistencies). Yamcs allows to use an algorithm which will perform the encoding without the addition of the extra argument:

```
<xtce:BinaryArgumentType name="barray">
    <xtce:AncillaryDataSet>
        <xtce:AncillaryData name="Yamcs">minLength=2</xtce:AncillaryData>
        <xtce:AncillaryData name="Yamcs">maxLength=10</xtce:AncillaryData>
    </xtce:AncillaryDataSet>
    <xtce:BinaryDataEncoding>
        <SizeInBits>
         <DynamicValue>
                <ParameterInstanceRef parameterRef="_yamcs_ignore" />
            </DynamicValue>
        </SizeInBits>
        <ToBinaryTransformAlgorithm name="LeadingSizeBinaryEncoder">
            <!-- the 16 passed to the constructor means the size is encoded on 16 bits -->
            <AlgorithmText language="java">
                org.yamcs.algo.LeadingSizeBinaryEncoder(16)
            </AlgorithmText>
        </ToBinaryTransformAlgorithm>
    </BinaryDataEncoding>
</BinaryArgumentType>
```

Note again the `<DynamicValue>` construct with a reference to `_yamcs_ignore` which will make yamcs ignore this section. The `<SizeInBits>` section can be removed from the file if XSD compliance is not important, Yamcs will not complain.

Note also the minLength and maxLength which are used to configure the minimum/maximum length of the accepted data (not including the 16 bits size tag!).

3.1.7 Absolute time data type

TBW

3.1.8 Enumerated data type

TBW

3.1.9 Aggregate data type

TBW

3.1.10 Array data type

TBW

3.2 Parameter Definitions

3.3 Container Definitions

Containers are the equivalent of packets in the usual terminology.

A container employs two mechanism to overcome limitations of the traditional “packet with parameters” approach. These mechanisms are *aggregation* and *inheritance*.

3.3.1 Container Aggregation

A container contains sequence entries which can be of two types:

1. **Parameter entries** pointing to normal parameters.
2. **Container entries** pointing to other containers which are then included in the big container.

Special attention must be given to the specification of positions of entries in the container. For performance reasons, it is preferable that all positions are absolute (i.e. relative to the beginning of the container) rather than relative to the previous entry. The Excel spreadsheet loader tries to transform the relative positions specified in the spreadsheet into absolute positions.

However, due to entries which can be of variable size, the situation cannot always be avoided. When an entry whose position is relative to the previous entry is subscribed, Yamcs adds to the subscription all the previous entries until it finds one whose position is absolute.

If an entry’s position depends on another entry (it can be the same in case the entry repeats itself) which is a Container Entry (i.e. makes reference to a container), and the referenced container doesn’t have the size in bits specified, then all the entries of the referenced container plus all the inheriting containers and their entries recursively are added to the subscription. Thus, the processing of this entry will imply the extraction of all parameters from the referenced container and from the inheriting containers. The maximum position reached when extracting entries from the referenced and inheriting containers is considered the end of this entry and used as the beginning of the following one.

3.3.2 Container Inheritance

Containers can point to another container through the baseContainer property, meaning that the baseContainer is extended with additional sequence entries. The inheritance is based on a condition put on the parameters from the baseContainer (e.g. a EDR_HK packet is a CCSDS packet with `apid=943` and `packetid=0x1300abcd`).

3.3.3 Little Endian Parameter Encoding

Yamcs supports only little or big endian (XTCE allows in addition arbytrary byte orders, this is not supported).

For little endian parameters which occupy a non-integer number of bytes, the following algorithm is applied to extract the parameter from the packet:

1. Based on the location of the first bit and on the size in bits of the parameter, find the sequence of bytes that contains the parameter. Only parameters that occupy at most 4 bytes are supported.
2. Read the bytes in reverse order in a 4 bytes int variable.

3. Apply the mask and the shift required to bring the parameter to the rightmost bit.

For example, assume this C struct on an x86 CPU:

```c
struct {
    unsigned int parameter1:4;
    unsigned int parameter2:16;
    unsigned int parameter3:12;
} x;
x.a=0x1;
x.b=0x2345;
x.c=0x678;
```

When converted to network order, this would give the sequence of hex bytes *51 34 82 67*. Thus, the definition of this packet should look like:

Parameter	Location	Size
parameter1	4	4
parameter2	4	16
parameter3	16	12

3.4 Alarm Definitions

Yamcs supports the XTCE notion of *alarms*. Based on the value of a parameter, Yamcs assigns a monitoring result to each parameter. The default monitoring result is DISABLED.

For enumerated parameters, the monitoring result can be:

- *null* (no alarm states are defined for this parameter)
- DISABLED (no alarms are applicable given the current set of updated parameter values)
- IN_LIMITS (an alarm was checked, but the value is within limits)
- WATCH
- WARNING
- DISTRESS
- CRITICAL
- SEVERE

For numeric parameters, the monitoring result can be:

- *null* (no alarm ranges are defined for this parameter)
- DISABLED (no alarms are applicable given the current set of updated parameter values)
- IN_LIMITS (an alarm was checked, but the value is within limits)
- WATCH
- WARNING
- DISTRESS
- CRITICAL
- SEVERE

Numeric parameter values get also tagged with a range condition LOW or HIGH, indicating whether the parameter value is too low or too high.

As part of the Mission Database definition, each parameter can have conditions by which the monitoring result should be set to a certain value. If the alarm conditions for multiple severity levels match, the highest severity level will always win.

For each parameter, multiple different sets of alarm conditions can be defined. A *context* condition is used to determine which set is applicable (for example, apply a different set of alarms if some other parameter is set to ‘CONTINGENCY MODE’).

3.5 Algorithm Definitions

Algorithms are user scripts that can perform arbitrary logic on a set of incoming parameters. The result is typically one or more derived parameters, called *output parameters*, that are delivered together with the original set of parameters (at least, if they have been subscribed to).

Output parameters are very much identical to regular parameters. They can be calibrated (in which case the algorithm’s direct outcome is considered the raw value), and they can also be subject to alarm generation.

Algorithms can be written in JavaScript, Python or Java. By default Yamcs supports JavaScript algorithms executed using the Nashorn JavaScript engine. Support for other languages (e.g. Python) requires installing additional dependencies.

Yamcs will bind these input parameters in the script’s execution context, so that they can be accessed from within there. In particular the following attributes and methods are made available:

value
the engineering value

rawValue
the raw value (if the parameter has a raw value)

monitoringResult
the result of the monitoring: *null*, `DISABLED`, `WATCH`, `WARNING`, `DISTRESS`, `CRITICAL` or `SEVERE`.

rangeCondition
If set, one of `LOW` or `HIGH`.

generationTimeMillis
The paramer generation time - milliseconds since Yamcs epoch.

aquisitionTimeMillis
The paramer acquisition time - milliseconds since Yamcs epoch.

generationTime()
The paramer generation time converted to Java Instant (by removing the leap seconds).

aquisitionTime()
The paramer acquisition time converted to Java Instant (by removing the leap seconds).

If there was no update for a certain parameter, yet the algorithm is still being executed, the previous value of that parameter will be retained.

3.5.1 Triggers

Algorithms can trigger on two conditions:

1. Whenever a specified parameter is updated
2. Periodically

Multiple triggers can be combined. In the typical example, an algorithm will trigger on updates for each of its input parameters. In other cases (for example because the algorithm doesn’t have any inputs), it may be necessary to trigger on some other parameter. Or maybe a piece of logic just needs to be run at regular time intervals, rather than with each parameter update.

If an algorithm was triggered and not all of its input parameters were set, these parameters *will* be defined in the algorithm's scope, but with their value set to `null`.

3.5.2 User Libraries

The Yamcs algorithm engine can be configured to import a number of user libraries. Just like with algorithms, these libraries can contain any sort of logic and are written in the same scripting language. Yamcs will load user libraries *one time only* at start-up in their defined order. This will happen before running any algorithm. Anything that was defined in the user library, will be accessible by any algorithm. In other words, user libraries define a kind-of global scope. Common use cases for libraries are: sharing functions between algorithms, shortening user algorithms, easier outside testing of algorithm logic, ...

Allowing to split the code in different user libraries is merely a user convenience. From the server perspective they could all be merged together in one big file.

3.5.3 Algorithm Scope

User algorithms have each their own scope. This scope is safe with respect to other algorithms (i.e. variables defined in algorithm *a* will not leak to algorithm *b*.

An algorithm's scope, however, is shared across multiple algorithm runs. This allows you to keep variables inside internal memory if needed. Do take caution with initializing your variables correctly at the beginning of your algorithm if you only update them under a certain set of conditions (unless of course you intend them to keep their value across runs).

3.5.4 Sharing State

If some kind of a shared state is required between multiple algorithms, the user libraries' shared scope could be used for this. In many cases, the better solution would be to just output a parameter from one algorithm, and input it into another. Yamcs will automatically detect such dependencies, and will execute algorithms in the correct order.

3.5.5 Historic Values

With what has been described so far, it would already be possible to store values in an algorithm's scope and perform windowing operations, such as averages. Yamcs goes a step further by allowing you to input a particular *instance* of a parameter. By default instance *0* is inputted, which means the parameter's actual value. But you could also define instance *-1* for inputting the parameter's value as it was on the previous parameter update. If you define input parameters for, say, each of the instances *-4*, *-3*, *-2*, *-1* and *0*, your user algorithm could be just a simple oneliner, since Yamcs is taking care of the administration.

Algorithms with windowed parameters will only trigger as soon as each of these parameters have all instances defined (i.e. when the windows are full).

3.5.6 JavaScript algorithms

The JavaScript algorithms are executed by the Nashorn engine. The engine is builtin the JDK versions 8 to 14; (it has been deprecated starting with Java 15 - should be replaced with one provided by the GraalVM project but this has not been tested).

The algorithm text is expected to contain the full function body. The body will be encapsulated in a javascript function like:

```
function algorithm_name(in_1, in_2, ..., out_1, out_2...) {
    <algorithm-text>
}
```

`in_x` and `out_x` are names assigned to the inputs/outpus in the algorithm definition.

The method can make use of the input variables and assign out_x.value (this is the engineering value) or out_x.rawValue (this is the raw value) and out_x.updated for each output variable.

The <out>.updated can be set to false to indicate that the output value has not to be further processed even if the algorithm has run. By default it is true - meaning that each time the algorithm is run, it is assumed that it updates all the output variables.

If out_x.rawValue is set and out_x.value is not, then Yamcs will run a calibration to compute the engineering value.

Note that some algorithms (e.g. command verifiers) need to return a value.

3.5.7 Python algorithms

This works very similarly with the JavaScript algorithms. The thing to pay attention is the indentation. The algorithm text wihch is specified in the spreadsheet will be automatically indented with 4 characters:

```
function algorithm_name(in_1, in_2, ..., out_1, out_2...) {
    <algorithm-text>
}
```

3.5.8 Java expression algorithms

This works similarly with the JavaScript and Python algorithms: a java class is generated containing the user defined algorithm text. It offers better peformance than the scripting algorithms because no script engine is involved.

```
... imports
... class declaration
private void execute_java_expr(ParameterValue input0, ParameterValue input1..., ParameterValue output0,␣
↪ParameterValue output1...) {
    <algorithm-text>
}
```

The first variables are the inputs, followed by the outputs. The java classe org.yamcs.parameter.ParameterValue[18] has to be used to get the values of the inputs (e.g. `getEngValue()` will give the engineering value) and set the value of the outputs. For example the text to add two inputs `pv0` and `pv1` into `AlgoFloatAdditionJe` could be:

```
float f0 = pv0.getEngValue().getFloatValue();
float f1 = pv1.getEngValue().getFloatValue();
AlgoFloatAdditionJe.setFloatValue(f0 + f1);
```

The `getFloatValue()` in the code above is because the engineering type is Float with sizeInBits=32. If the wrong get is used on a org.yamcs.parameter.Value[19], an exceptio will be thrown by the algorithm (should be visible in the yamcs-web as well as in the logs).

The algorithm can leave the output values unset; in that case the values will not be used further.

In case the algorithm is used for a command verifier (see below), it has to return a value. A boolean value of `true` (in fact java `Boolean.TRUE` object) means that the verifier has succeeded, `null` means that the verifier is still pending. Any other value means that the verifier has failed; the object will be converted to string and used as an explanation for the failure.

[18] https://yamcs.org/javadoc/yamcs/org/yamcs/parameter/ParameterValue.html

[19] https://yamcs.org/javadoc/yamcs/org/yamcs/parameter/Value.html

3.5.9 Java algorithms

The algorithm text is a class name with optionally parantheses enclosed string that is parsed into an object by a yaml parser. Unlike the java-expression algorithms, the Java algorithms require the user to pre-compile the classes into a jar and place it on the server in the lib/ext directory.

Yamcs will locate the given class which must be implementing the org.yamcs.algorithms.AlgorithmExecutor[20] interface and will create an object with a constructor with three parameters:

```
MyAlgorithmExecutor(Algorithm algorithmDef, AlgorithmExecutionContext context, Object arg)
```

- `algorithmDef` represents the algorithm definition; it can be used for example to retrieve the MDB algorithm name, input parameters, etc.
- `context` is an object holiding some contextual information related to where the algorithm is running. Generally this refers to a processor but for command verifiers there is a restricted context to distinguish the same algorithm running as verifier for different commands.
- `arg` is an optional argument parsed using the snakeyaml parser (can be a Integer, Long, Double, Map or List).

If the optional argument is not present in the algorithm text definition, then the class constructor should only have two parameters.

The class has two main methods `updateParameters` which is called each time one of input parameters changes and `runAlgorithm` which runs the algorithm and returns the output values. The algorithm is free to chose which output values are returned at each run (it could also return an empty list when no value has been generated).

The abstract class org.yamcs.algorithms.AbstractAlgorithmExecutor[21] offers some helper methods and can be used as base class for implementation of such algorithm.

If the algorithm is used for data decoding, it has to implement the org.yamcs.mdb.DataDecoder[22] interface instead (see below).

3.5.10 Command verifier algorithms

Command verifier algorithms are special algorithms associated to the command verifiers. Multiple instances of the same algorithm may execute in parallel if there are multiple pending commands executed in parallel.

These algorithms are special as they can use as input variables not only parameters but also command arguments and command history events. These are specified by using "/yamcs/cmd/arg/" and "/yamcs/cmdHist" prefix respectively.

In addition these algorithms have to return a boolean value (whereas the normal algorithms only have to write to output variables). The returned value is used to indicate if the verifier has succeeded or failed. No return value will mean that the verifier is still pending.

3.5.11 Data Decoding algorithms

The Data Decoding algorithms are used to extract a raw value from a binary buffer. These algorithms do not produce any output and are triggered whenever the parameter has to be extracted from a container.

These algorithms work differently from the other ones and have are some limitations:

- only Java is supported as a language
- not possible to specify input parameters

[20] https://yamcs.org/javadoc/yamcs/org/yamcs/algorithms/AlgorithmExecutor.html
[21] https://yamcs.org/javadoc/yamcs/org/yamcs/algorithms/AbstractAlgorithmExecutor.html
[22] https://yamcs.org/javadoc/yamcs/org/yamcs/mdb/DataDecoder.html

These algorithms have to implement the interface org.yamcs.mdb.DataDecoder[23].

3.6 Command Definitions

3.7 Loading TM/TC Definitions

3.7.1 XTCE Loader

This loader reads TM/TC definitions from an XML file compliant with the XTCE exchange format coordinated by OMG. The Yamcs database is very close to XTCE, which makes this mapping relatively straightforward. For more information about XTCE, see http://www.xtce.org.

3.7.1.1 Configuration

The loader is configured in `etc/mdb.yaml` or in the instance configuration by specifying the 'type' as `xtce`, and providing the location of the XML file in the `file` attribute.

```
- type: "xtce"
  args:
    file: "BogusSAT.xml"
    autoTmPartitions: true
    #fileset: ["a*.xml", "b.xml"]
```

Configuration Options

file (string)
The filename to be loaded. Either this or the `fileset` attribute are required

fileset (string or list of strings)
Available since Yamcs 5.2; can be used to load multiple XML files. A glob pattern can be used to match multiple files and/or the files can be specified in the list. The `**` for matching directories recursively is not supported. If the `fileset` option is used, the `subLoader` cannot be used to load child subsystems. This is because it is not possible to specify which subsystem will be the parent of the child.

autoTmPartitions (boolean)
If true, Yamcs will automatically mark to be used as archive partitions all containers which do not have a parent. If this option is false, the containers can still be manually marked by using the ancillary data property `UseAsArchivingPartition`:

```
</xtce:SequenceContainer>
  ....
  <xtce:AncillaryDataSet>
      <xtce:AncillaryData name="Yamcs">
          UseAsArchivingPartition
      </xtce:AncillaryData>
   </xtce:AncillaryDataSet>
</xtce:SequenceContainer>
```

Default: true

3.7.1.2 Compatibility

Yamcs does not seek full compliance with XTCE. It only reads the parts that relate to concepts in its internal Mission Database. This chapter presents an overview of the unsupported features and details where the implementation differs from the standard.

[23] https://yamcs.org/javadoc/yamcs/org/yamcs/mdb/DataDecoder.html

Note that when reading the XML XTCE file Yamcs is on purpose tolerant, it ignores the tags it does not know and it also strives to be backward compatible with XTCE 1.0 and 1.1. Thus the fact that an XML file loads in Yamcs does not mean that is 100% valid. Please use a generic XML validation tool or the xtcetools[24] project to validate your XML file.

The following concepts are *not supported*:

- `Stream` - data is assumed to be injected into Yamcs as packets, any stream processing has to be done as part of the data link definition and is not based on XTCE.
- `Message`
- `ParameterSegmentRefEntry`
- `ContainerSegmentRefEntry`
- `DiscreteLookupList`
- `ErrorDetectCorrectType`. Note that error detection/correction is implemented directly into the Yamcs data links.
- `ContextSignificanceList`
- `ParameterToSetList`
- `ParameterToSuspendAlarmsOnSet`
- `RestrictionCriteria/NextContainer`

The other elements are supported one way or another, exceptions or changes from the specs are given in the sections below.

Header

- Only the version and date are supported. `AuthorSet` and `NoteSet` are ignored.

Data Encodings

- changeThreshold
  Not supported.
- FromBinaryTransformAlgorithm
  In XTCE the `FromBinaryTransformAlgorithm` can be specified for the `BinaryDataEncoding`. It is not clear how exactly that is supposed to work. In Yamcs the `FromBinaryTransformAlgorithm` can be specified on any `XyzDataEncoding` and is used to convert from binary to the raw value which is supposed to be of type Xyz.
- ToBinaryTransformAlgorithm
  not supported for any data encoding
- FloatDataEncoding
  Yamcs supports IEEE754_1985, MILSTD_1750A and STRING encoding. STRING is not part of XTCE - if used, a StringDataEncoding can be attached to the FloatDataEncoding and the string will be extracted according to the StringDataEncoding and then parsed into a float or double according to the sizeInBits of FloatDataEncoding. DEC, IBM and TI encoding are not supported.
- StringDataEncoding
  For variable size strings whose size is encoded in front of the string, Yamcs allows to specify only for command arguments sizeInBitsOfSizeTag = 0. This means that the value of the argument will be inserted without providing the information about its size. The receiver has to know how to derive the size. This has been implemented for compatibility with other systems (e.g. SCOS-2k) which allows this - however it is not allowed by XTCE which enforces sizeInBitsOfSizeTag > 0.

[24] https://gitlab.com/dovereem/xtcetools

Data Types

- ValidRangeSet
  Introduced in XTCE 1.2 for command arguments. Yamcs only supports one range in the set.
- BooleanDataType
  In XTCE, each `BooleanDataType` has a string representation. In Yamcs the value is mapped to a org.yacms.parameter.BooleanValue or the protobuf equivalent that is a wrapper for a boolean (either true or false in all sane programming languages). The string value is neverhteless supported in comparisons and mathalgorithms but they are converted internally to the boolean value. If you want to get to the string representation from the client, use an `EnumeratedParameterType`.
- RelativeTimeDataType
  Not supported.

Monitoring

- ParameterSetType
  `parameterRef` is not supported. According to XTCE doc this is "Used to include a Parameter defined in another sub-system in this sub-system". It is not clear what it means "to include". Parameters from other space systems can be referenced using a fully qualified name or a relative name.
- ParameterProperties
  `PhysicalAddressSet`, `SystemName` and `TimeAssociation` are not supported.
- Containers
  `BinaryEncoding` not supported in the container definitions.
- StringParameterType
  Alarms are not supported.

Commanding

- Arrays are not supported for commands (they are for telemetry).
- ArgumentRefEntry
  `IncludeCondition` and `RepeatEntry` are not supported.
- Multiple CompleteVerifiers can be declared but the success of any of them will make the command complete successfully; XTCE specifies that all of them have to succeed for the command to be declared successful.
  Note that when a command is completed (with success or failure), all the pending verifies are canceled. This means that if mutiple CompleteVerifiers are declared, the first one finishing will decide the outcome of the command.

Algorithms

- `OnContainerUpdateTrigger` is not supported.

3.7.2 Spreadsheet Loader

3.7.2.1 Configuration

The loader is configured in `etc/mdb.yaml` or in the instance configuration by specifying the 'type' as `sheet`, and providing the location of the XML file in the `spec` attribute.

```yaml
- type: "sheet"
  spec: "BogusSAT.xls"
```

3.7.2.2 Conventions

Multiple Space Systems

A tree of multiple space systems can be defined in a single Excel file.

To define the space system hierarchy, the convention is that all the sheets that do not have a prefix contain data for the main space system whose name is defined in the *General Sheet* (page 30). To define data in subsystems, a syntax like `SYSTEM1|SYSTEM2|Containers` can be used. This definition will create a SYSTEM1 as part of the main space system and a child SYSTEM2 of SYSTEM1. Then the containers will be loaded in SYSTEM2.

The spreadsheet loader scans and creates the subsystem hierarchy and then it loads the data inside the systems traversing the hierarchy in a depht-first order.

Number Base

Numeric values can be entered as decimals or as hexadecimals (with prefix `0x`)

Referencing Parameter and Containers

Each time a name reference is mentioned in the spreadsheet, the following rules apply:

- The reference can use UNIX like directory access expressions, such as `../a/b`.
- If the name is not found as a qualified parameter, and the option enableAliasReferences is configured for the SpreadsheetLoader, the parameter is looked up through all the aliases of the parent systems.

The result of the lookup depends on the exact tree configuration in `mdb.yaml`

3.7.2.3 General Sheet

This sheet is required.

- **format version**

  Used by the loader to ensure a compatible spreadsheet structure

- **name**

  Name of the MDB

- **document version**

  Used by the author to track versions in an arbitrary manner

3.7.2.4 Containers Sheet

The sheet contains description of the content of the container (packet). As per XTCE, a container is a structure describing a binary chunk of data composed of multiple entries.

A container can inherit from other container - meaning that it takes all entries from the parent and it adds some more. It can have two types of entries:

- parameters
- other containers (this is called aggregation)

General conventions:

- first line with a new 'container name' starts a new packet
- second line after a new 'container name' should contain the first measurement
- empty lines are only allowed between two packets

Comment lines starting with # on the first column can appear everywhere and are ignored.

- **container name**

  The relative name of the packet inside the space system

- **parent**

  Parent container and position in bits where the subcontainer starts, for example `PARENT_CONTAINER:64`. If position in bits is not specified, the default position is to start from the last parameter in the parent. If parent is not specified, either the container is the root, or it can be used as part of another container in aggregation.

- **condition**

  Inheritance condition, usually specifies a switch within the parent which activates this child, for example *MID=0x101* There are currently three forms supported:

  - Simple condition: `Parameter==value`
  - Condition list: `Parameter==value;Parameter2==value2` - all conditions must be true
  - Boolean condition: `op(epx1;exp2;...;expn)`
    - op is '&' (AND) or '|' (OR)
    - expi is a boolean expression or a simple condition

  Currently the only supported conditions are on the parameters of the parent container. This cover the usual case where the parent defines a header and the inheritance condition is based on paraemters from the header.

3.7.2.5 Parameters Sheet

This sheet contains parameter information.

This sheet may be named:

- **Parameters** for telemetered parameters.
- **DerivedParameters** for parameters that are the results of algorithm computations.
- **LocalParameters** for parameters that are local to Yamcs and that can be set by users.

A parameter when extracted from a binary packet has two forms: a raw value and an engineering value. The extraction from the raw packet is performed according to the encoding, whereas the conversion from raw to engineering value is performed by a calibrator. This sheet can also be used to specify parameters without encoding - if they are received already extracted, Yamcs can do only their calibration. Or it can be that a parameter is already calibrated, it can still be specified here to be able to associate alarms.

Empty lines can appear everywhere and are ignored. Comment lines starting with # on the first column can appear everywhere and are ignored.

- **name**

  The name of the parameter in the namespace.

- **encoding**

  Description on how to extract the raw type from the binary packet. See below for all supported encodings.

- **raw type**

  Documented below.

- **eng type**

  Documented below.

- **eng unit**

  Free-form textual description of unit(s). E.g. degC, W, V, A, s, us

- **calibration**

  Name of a calibration described in the Calibration sheet, leave empty if no calibration is applied

- **description**

  Optional human-readable text

- **namespace:<NS-NAME>**

If present, these columns can be used to assign additional names to the parameters in the namespace NS-NAME. Any number of columns can be present to give additional names in different namespaces.

3.7.2.5.1 Encoding and Raw Types The raw type and encoding describe how the parameter is encoded in the binary packet. All types are case-insensitive.

Unsigned Integers Raw type: `uint`

Encoding:

Encoding	Description
`unsigned(<n>,<BE|LE>)`	unsigned integer
`<n>`	shortcut for `unsigned(<n>,BE)`

Where:

- `n` is the size in bits
- `LE` = little endian
- `BE` = big endian

Signed Integers Raw type: `int`

Encoding:

Encoding	Description
`twosComplement(<n>,<BE|LE>)`	two's complement encoding
`signMagnitude(<n>,<BE|LE>)`	sign magnitude encoding - first (or last for LE) bit is the sign, the remaining bits represent the magnitude (absolute value).
`<n>`	shortcut for `twosComplement(<n>,BE)`

Where:

- `n` is the size in bits
- `LE` = little endian
- `BE` = big endian

Floats Raw type: `float`

Encoding:

Encoding	Description
`ieee754_1985(<n>,<BE|LE>)`	IEE754_1985 encoding
`<n>`	shortcut for `ieee754_1985(<n>,BE)`

Where:

- `n` is the size in bits
- `LE` = little endian

- BE = big endian

Booleans Raw type: `boolean`

Encoding: Leave empty. 1 bit is assumed.

String Raw type: `string`

Encoding:

Encoding	Description
`fixed(<n>, <charset>)`	fixed size string. The string has to start at a byte boundary inside the container.
`PrependedSize(<x>, <charset>)`	string whose length in bytes is specified by the first `x` bits of the array
`<n>`	shortcut for `fixed(<n>)`
`terminated(<0xBB>, <charset><m>)`	terminated string

Where:

`n` is the size in bits. Only multiples of 8 are supported.

`x` is the size in bits of the size tag. Only multiples of 8 are supported. The size must be expressed in bytes.

`charset` is one of the charsets supported by java[25] (UTF-8, ISO-8859-1, etc). Default: UTF-8.

`0xBB` specifies a byte that is the string terminator. Pay attention to the parameters following this one; if the terminator is not found the entire buffer will be consumed.

Binary Raw type: `binary`

Encoding:

Encoding	Description
`fixed(<n>)`	fized size byte array
`PrependedSize(<x>)`	byte array whose size in bytes is specified in the first `x` bits of the array
`<n>`	shortcut for `fixed(<n>)`

Where:

`n` is the size in bits. Only multiples of 8 are supported and it has to start at a byte boundary.

`x` is the size in bits of the size tag. Note that while `x` can be any number <= 32, the byte array has to start at a byte boundary.

Custom Raw type: *any*

Encoding: `custom(<n>,algorithm)`

The decoding will be performed by a user defined algorithm.

- `<n>` is optional and may be used to specify the size in bits of the entry in the container (in case the size is fixed) - it is used for optimizing the access to the parameters following this one.

[25] https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html

- `algorithm` the name of the algorithm - it has to be defined in the *Algorithms* sheet

3.7.2.5.2 Engineering Types Engineering types describe a parameter in its processed form (i.e. after any calibrations). All types are case-insensitive.

Depending on the combination of raw and engineering type, automatic conversion is applicable. For more advanced use cases, define and refer to a Calibrator in the Calibration Sheet

Type	Description	Automatic Conversion
uint	Unsigned 32 bit integer - it corresponds to `int` in java and `uint32` in protobuf	From `int`, `uint` or `string`
uint64	Unsigned 64 bit integer - it corresponds to `long` in java and `uint64` in protobuf	From `int`, `uint` or `string`
int	Signed 32 bit integer - it corresponds to `int` in java and `int32` in protobuf	From `int`, `uint` or `string`
int64	Signed 64 bit integer - it corresponds to `long` in java and `int64` in protobuf	From `int`, `uint` or `string`
string	Character string - it corresponds to `String` in java and `string` in protobuf	From `string`
float	32 bit floating point number - it corresponds to `float` in java and protobuf	From `float`, `int`, `uint` or `string`
double	64 bit floating point number - it corresponds to `double` in java and protobuf	From `float`, `int`, `uint` or `string`
enumerated	A kind of string that can only be one out of a fixed set of predefined state values. It corresponds to `String` in java and `string` in protobuf.	From `int` or `uint`. A Calibrator is required.
boolean	A binary true/false value - it corresponds to 'boolean' in java and 'bool' in protobuf	From any raw type. Values equal to zero, all-zero bytes or an empty string are considered *false*.
binary	Byte array - it corresponds to `byte[]` in java and `bytes` in protobuf.	From `bytestream` only

3.7.2.6 Calibration Sheet

This sheet contains calibration data including enumerations.

calibrator name
Name of the calibration - it has to match the calibration column in the Parameter sheet.

type One of the following:

- `polynomial` for polynomial calibration. Note that the polynomial calibration is performed with double precision floating point numbers even though the input and/or output may be 32 bit.
- `spline` for linear spline (pointpair) interpolation. As for the polynomial, the computation is performed with double precision numbers.
- `enumeration` for mapping enumeration states.
- `java-expression` for writing more complex functions.

calib1

- If the type is `polynomial`: it list the coefficients, one per row starting with the constant and up to the highest grade. There is no limit in the number of coefficients (i.e. order of polynomial).
- If the type is `spline`: start point (x from (x,y) pair)
- If the type is `enumeration`: numeric value
- If the type is `java-expression`: the textual formula to be executed (see below)

calib2

- If the type is `polynomial`: leave *empty*
- If the type is `spline`: stop point (y) corresponding to the start point(x) in `calib1`
- If the type is `enumeration`: text state corresponding to the numeric value in `calib1`
- If the type is `java-expression`: leave *empty*

3.7.2.6.1 Java Expressions

This is intended as a catch-all case. XTCE specifies a MathOperationCalibration calibrator that is not implemented in Yamcs. However these expressions can be used for the same purpose.

They can be used for float or integer calibrations.

The expression appearing in the *calib1* column will be enclosed and compiled into a class like this:

```
package org.yamcs.mdb.jecf;
public class Expression665372494 implements org.yamcs.mdb.CalibratorProc {
    public double calibrate(double rv) {
            return <expression>;
    }
}
```

The expression has usually to return a double; but java will convert implicitly any other primitive type to a double.

Java statements cannot be used but the conditional operator `? :` can be used; for example this expression would compile fine:

```
rv>0?rv+5:rv-5
```

Static functions can be also referenced. In addition to the usual Java ones (e.g. Math.sin, Math.log, etc) user own functions (that can be found as part of a jar on the server in the lib/ext directory) can be referenced by specifying the full class name:

```
my.very.complicated.calibrator.Execute(rv)
```

3.7.2.7 Algorithms Sheet

This sheet contains arbitrarily complex user algorithms that can set (derived) output parameters based on any number of input parameters.

Comment lines starting with “#” on the first column can appear everywhere and are ignored. Empty lines are used to separate algorithms and cannot be used inside the specification of one algorithm.

algorithm name
The identifying name of the algorithm.

algorithm language
The programming language of the algorithm. Currently supported values are JavaScript, python and Java.

text The code of the algorithm (see below for how this is interpreted).

trigger

Optionally specify when the algorithm should trigger:

- `OnParameterUpdate('/some-param', 'some-other-param')` Execute the algorithm whenever *any* of the specified parameters are updated
- `OnInputParameterUpdate` This is the same as above for all input parameters (i.e. execute whenever *any* input parameter is updated).
- `OnPeriodicRate(<fireRate>)` Execute the algorithm every `fireRate` milliseconds
- `none` The algorithm doesn't trigger automatically but can be called upon from other parts of the system (like the command verifier)

The default is none.

in/out

Whether a parameter is inputted to, or outputted from the algorithm. Parameters are defined, one per line, following the line defining the algorithm name

parameter reference

Algorithms can be interdependent, meaning that the output parameters of one algorithm could be used as input parameters of another algorithm.

instance

Allows inputting a specific instance of a parameter. At this stage, only values smaller than or equal to zero are allowed. A negative value, means going back in time. Zero is the default and means the actual value. This functionality allows for time-based window operations over multiple packets. Algorithms with windowed parameters will only trigger as soon as all of those parameters have all instances defined (i.e. when the windows are full).

Note that this column should be left empty for output parameters.

name used in the algorithm

An optional friendlier name for use in the algorithm. By default the parameter name is used, which may lead to runtime errors depending on the naming conventions of the applicable script language.

Note that a unique name is required in this column, when multiple instances of the same parameter are inputted.

See *Algorithm Definitions* (page 23)

3.7.2.8 Alarms Sheet

This sheet defines how the monitoring results of a parameter should be derived. E.g. if a parameter exceeds some pre-defined value, this parameter's state changes to `CRITICAL`.

parameter name

The reference name of the parameter for which this alarm definition applies

context

A condition under which the defined triggers apply. This can be used to define multiple different sets of triggers for one and the same parameter, that apply depending on some other condition (typically a state of some kind). When left blank, the defined set of conditions are assumed to be part of the *default* context.

Contextual alarms are evaluated from top to bottom, until a match is found. If no context conditions apply, the default context applies.

report

When alarms under the given context should be reported. Should be one of `OnSeverityChange` or `OnValueChange`. With `OnSeverityChange` being the default. The condition `OnValueChange` will check value changes based on the engineering values. It can also be applied to a parameter without any defined severity levels, in which case an event will be generated with every change in value.

minimum violations

Number of successive instances that meet any of the alarm conditions under the given context before the alarm event triggers (defaults to 1). This field affects when an event is generated (i.e. only after X violations). It does not affect the monitoring result associated with each parameter. That would still be out of limits, even after a first violation.

watch: trigger type

One of `low`, `high` or `state`. For each context of a numeric parameter, you can have both a low and a high trigger that lead to the `WATCH` state. For each context of an enumerated parameter, you can have multiple state triggers that lead to the `WATCH` state.

watch: trigger value

If the trigger type is `low` or `high`: a numeric value indicating the low resp. high limit value. The value is considered inclusive with respect to its nominal range. For example, a low limit of 20, will have a `WATCH` alarm if and only if its value is smaller than 20.

If the trigger value is `state`: a state that would bring the given parameter in its `WATCH` state.

warning: trigger type, warning: trigger value

Analogous to `watch` condition

distress: trigger type, distress: trigger value

Analogous to `watch` condition

critical: trigger type, critical: trigger value

Analogous to `watch` condition

severe: trigger type, severe: trigger value

Analogous to `watch` condition

3.7.2.9 Commands Sheet

This sheet contains commands description, including arguments. General convention:

- First line with a new ‘Command name’ starts a new command
- Second line after a new ‘Command name’ should contain the first command arguments
- Empty lines are only allowed between two commands.

Command name

The name of the command. Any entry starting with # is treated as a comment row

parent

name of the parent command if any.

Can be specified starting with / for an absolute reference or with ../ for pointing to parent SpaceSystem :x means that the arguments in this container start at position x (in bits) relative to the topmost container. Currently there is a problem for containers that have no argument: the bit position does not apply to children and has to be repeated.

argAssignment

name1=value1;name2=value2.. where name1,name2.. are the names of arguments which are assigned when the inheritance takes place

flags

For commands: A=abstract. For arguments: L = little endian

argument name

From this column on, most of the cells are valid for arguments only. These have to be defined on a new row after the command. The exceptions are: description, aliases

relpos

Relative position to the previous argument. Default: 0

encoding
How to convert the raw value to binary. The supported encodings are listed in the table below.

eng type
Engineering type; can be one of: uint, int, float, string, binary, enumerated, boolean or FixedValue. FixedValue is like binary but is not considered an argument but just a value to fill in the packet.

raw type
Raw type: one of the types defined in the table below.

(default) value
Default value. If eng type is FixedValue, this has to contain the value in hexadecimal. Note that when the size of the argument is not an integer number of bytes (which is how hexadecimal binary strings are specified), the most significant bits are ignored.

eng unit

calibration
Point to a calibration from the Calibration sheet

range low
The value of the argument cannot be smaller than this. For strings and binary arguments this means the minimum length in characters, respectively bytes.

range high
The value of the argument cannot be higher than this. Only applies to numbers. For strings and binary arguments this means the minimum length in characters, respectively bytes.

description
Optional free text description

3.7.2.9.1 Encoding and Raw Types for command arguments The raw type and encoding describe how the argument is encoded in the binary packet. All types are case-insensitive.

Unsigned Integers Raw type: `uint`

Encoding:

Encoding	Description
`unsigned(<n>,<BE|LE>)`	unsigned integer
`<n>`	shortcut for `unsigned(<n>,BE)`

Where:

- `n` is the size in bits
- `LE` = little endian
- `BE` = big endian

Signed Integers Raw type: `int`

Encoding:

Encoding	Description
`twosComplement(<n>, <BE|LE>)`	two's complement encoding
`signMagnitude(<n>,<BE|LE>)`	sign magnitude encoding - first (or last for LE) bit is the sign, the remaining bits represent the magnitude (absolute value).
`<n>`	shortcut for `twosComplement(<n>,BE)`

Where:

- `n` is the size in bits
- `LE` = little endian
- `BE` = big endian

Floats Raw type: `float`

Encoding:

Encoding	Description
`ieee754_1985(<n>,<BE|LE>)`	IEE754_1985 encoding
`<n>`	shortcut for `ieee754_1985(<n>,BE)`

Where:

- `n` is the size in bits
- `LE` = little endian
- `BE` = big endian

Booleans Raw type: `boolean`

Encoding: Leave empty. 1 bit is assumed.

String Raw type: `string`

Encoding:

Encoding	Description
`fixed(<n>, <charset>)`	fixed size string
`PrependedSize(<x>, <charset><m>)`	string whose length in bytes is specified by the first `x` bits of the array
`<n>`	shortcut for `fixed(<n>)`
`terminated(<0xBB>, <charset><m>)`	terminated string

Where:

`n` is the size in bits. Only multiples of 8 are supported.

`x` is the size in bits of the size tag. Only multiples of 8 are supported. The size must be expressed in bytes.

`charset` is one of the charsets supported by java[26] (UTF-8, ISO-8859-1, etc). Default: UTF-8.

[26] https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html

`m` if specified, it is the minimum size in bits of the encoded value. Note that the size reflects the real size of the string even if smaller than this minimum size. This option has been added for compatibility with the Airbus CGS system but its usage is discouraged since it is not compliant with XTCE.

`0xBB` specifies a byte that is the string terminator.

Binary Raw type: `binary`

Encoding:

Encoding	Description
`fixed(<n>)`	fized size byte array
`PrependedSize(<x>)`	byte array whose size in bytes is specified in the first `x` bits of the array
`<n>`	shortcut for `fixed(<n>)`

Where:

`n` is the size in bits. Only multiples of 8 are supported and it has to start at a byte boundary.

`x` is the size in bits of the size tag. Note that while `x` can be any number <= 32, the byte array has to start at a byte boundary.

3.7.2.10 CommandOptions Sheet

This sheet defines two types of options for commands:

- transmission constraints - these are conditions that have to be met in order for the command to be sent.
- command significance - this is meant to flag commands that have a certain significance. Currently the significance is only used by the end user applications to raise the awareness of the operator when sending such command.

Command name

The name of the command. Any entry starting with # is treated as a comment row

Transmission Constraints

Constrains can be specified on multiple lines. All of them have to be met for the command to be allowed for transmission.

Constraint Timeout

This refers to the left column. A command stays in the queue for that many milliseconds. If the constraint is not met, the command is rejected. 0 means that the command is rejected even before being added to the queue, if the constraint is not met.

Command Significance

Significance level for commands. Depending on the configuration, an extra confirmation or certain privileges may be required to send commands of high significance. One of:

- none
- watch
- warning
- distress
- critical
- severe

Significance Reason
A message that will be presented to the user explaining why the command is significant.

3.7.2.11 CommandVerification Sheet

The Command verification sheet defines how a command shall be verified once it has been sent for execution.

The transmission/execution of a command usual goes through multiple stages and a verifier can be associated to each stage. Each verifier runs within a defined time window which can be relative to the release of the command or to the completion of the previous verifier. The verifiers have three possible outcomes:

- **OK**
  the stage has been passed successfully.
- **NOK**
  the stage verification has failed (for example there was an error on-board when executing the command, or the uplink was not activated).
- **timeout**
  the condition could not be verified within the defined time interval.

For each verifier it has to be defined what happens for each of the three outputs.

Command name
The command relative name as defined in the Command sheet. Referencing commands from other subsystems is not supported.

CmdVerifier Stage
Any name for a stage is accepted but XTCE defines the following ones:

- TransferredToRange
- SentFromRange
- Received
- Accepted
- Queued
- Execution
- Complete
- Failed

Yamcs interprets these as strings without any special semantics. If special actions (like declaring the command as completed) are required for Complete or Failed, they have to be configured in OnuSccess/OnFail/OnTimeout columns. By default command history events with the name Verification_<stage> are generated.

CmdVerifier Type
Supported types are:

- container – the command is considered verified when the container is received. Note that this cannot generate a Fail (NOK) condition - it's either OK if the container is received in the timewindow or timeout if the container is not received.
- algorithm – the result of the algorithm run is used as the output of the verifier. If the algorithm is not run (because it gets no inputs) or returns null, then the timeout condition applies

CmdVerifier Text
Depending on the type:

- container: is the name of the container from the Containers sheet. Reference to containers from other space systems is not supported.

- algorithm: is the name of the algorithm from the Algorithms sheet. Reference to algorithms from other space systems is not supported.

Time Check Window
start,stop in milliseconds defines when the verifier starts checking the command and when it stops.

checkWindow is relative to

- LastVerifier (default) – the start,stop in the window definition are relative to the end of the previous verifier. If there is no previous verifier, the start,stop are relative to the command release time. If the previous verifier ends with timeout, this verifier will also timeout without checking anything.
- CommandRelease - the start,stop in the window definition are relative to the command release.

OnSuccess
Defines what happens when the verification returns true. It has to be one of:

- SUCCESS: command considered completed successful (CommandComplete event is generated)
- FAIL: CommandFailed event is generated
- none (default) – only a Verification_stage event is generated without an effect on the final execution status of the command.

OnFail
Same like OnSuccess but the evnet is generated in case the verifier returns false.

OnTimeout
Same as OnSuccess but the event is generated in case the verifier times out.

3.7.2.12 ChangeLog Sheet

This sheet contains the list of the revision made to the described space system.

The spreadsheet loader loads TM/TC definitions from an Excel spreadsheet. The spreadsheet structure must follow a specific structure. The advantage of this loader is that the Excel files are very convenient to modify with any spreadsheet program. It is recommended to start from an existing example and replace its content as required.

The Excel file must be in Excel 97-2003 Format (`.xls`). `.xlsx` is not supported.

3.7.3 Empty Node

This loader allows to create an empty node in the space system hierarchy with a given name.

For example this configuration will create two parallel nodes `/N1` and `/N2` and underneath each of them, load the xls files of the simulator.

```yaml
mdb:
  - type: "emptyNode"
    spec: "N1"
    subLoaders:
      - type: "sheet"
        spec: "mdb/simulator-ccsds.xls"
        subLoaders:
          - type: "sheet"
            spec: "mdb/landing.xls"

  - type: "emptyNode"
    spec: "N2"
    subLoaders:
      - type: "sheet"
        spec: "mdb/simulator-ccsds.xls"
        subLoaders:
```

(continues on next page)

(continued from previous page)

```
          - type: "sheet"
            spec: "mdb/landing.xls"
```

Yamcs constructs its Mission Database on server startup from a configurable tree of *loaders*. Each loader is responsible for a particular space system, and optionally its sub-space systems. It is not possible for one loader to add to adjacent space systems.

Configuration Example

```
simulator:
  - type: "sheet"
    spec: "mdb/simulator-ccsds.xls"
    subLoaders:
      - type: "sheet"
        spec: "mdb/simulator-tmtc.xls"
```

Multiple different types of loaders may be combined in the loader tree to assemble the full mission database. Each loader can load definitions from any source as long as the definitions can be mapped into Yamcs internal database format, which is based on the XTCE constructs.

For start-up performance, the database is cached serialized on disk in the cache directory. The cached database is composed of two files, one storing the data itself and the other one storing the time when the cache file has been created. These files should be considered Yamcs internal and are subject to change.

Since Yamcs 5.2.0, a database loader (for example the XTCE loader) is able to load multiple space systems which will all be added as siblings. In this case, the subLoaders option cannot be anymore specified (because otherwise it would not be clear to which of the loaded space systems the children will be added).

Note: Yamcs does not persist TM/TC definitions and therefore does not have any “import” functionality.

4. Data Management

Yamcs contains a generic data management system that combines two fundamental principles:

- Managing **static tables** of data.
- Managing **continuous streams** of data.

Both concepts are combined in a unifying **Stream SQL** language.

In addition, Yamcs contains a **Parameter Archive** that is specifically optimized for retrieval of parameter values. The Parameter Archive contains derived data and can be rebuilt at any time from the static database tables.

4.1 Streams

The concept of *streams* was inspired from the domain of Complex Event Processing (CEP) or Stream Processing. Streams are similar to database tables, but represent continuously moving data. SQL-like statements can be defined on streams for filtering, aggregation, merging or other operations. Yamcs uses streams for distributing data between all components running inside the same JVM. The most important place where streams are used is to make the connection between the data links and processors.

Typically there is a stream for realtime telemetry called `tm_realtime`, one for realtime processed parameters called `pp_realtime`, one for commands called `tc`, etc.

At instance startup, Yamcs will automatically create all the standard streams specified in the streamConfig property.

```
streamConfig:
    tm:
        - name: "tm_realtime"
          processor: "realtime"
        - name: "tm2_realtime"
          rootContainer: "/YSS/SIMULATOR/tm2_container"
          processor: "realtime"
        - name: "tm_dump"
    tc:
        - name: tc_sim
          processor: realtime
          tcPatterns: ["/YSS/SIMULATOR/.*"]
        - name: tc_tse
          processor: realtime
    invalidTm: "invalid_tm_stream"
    cmdHist: ["cmdhist_realtime", "cmdhist_dump"]
    event: ["events_realtime", "events_dump"]
    param: ["pp_realtime", "pp_tse", "sys_param", "proc_param"]
    parameterAlarm: ["alarms_realtime"]
    eventAlarm: ["event_alarms_realtime"]
    sqlFile: "etc/extra_streams.sql"
```

The configuration contains an entry for each default stream type:

tm (list)

contains a list of TM streams. Each stream has an mandatory name, and an optional processor and rootContainer properties. The processor property is used to attach the stream to a specific processor. If no processor is specified, the stream can still be used for example for recording the data in the archive - this is typical for a dump stream that retrieves non realtime data. The rootContainer property specifies which XTCE container shall be used for processing the packets on this stream.

tc (list)

contains a list of TC streams. Each stream has a mandatory name and an optional processor and tcPatterns properties. The processor is used to attach the stream to a specific processor. If no processor is specified, the stream can be used by other services. For example the CFDP service will push the CFDP PDUs to a stream from which they can be copied to a TC stream using some sql commands (as demonstrated in the cfdp example). The tcPatterns property is used to determine which command will be sent via this stream. It contains a list of regular expressions which are matched against eh command fully qualified name. If the patterns are not specified, it means that all commands will match. The ordering of the streams in this list is important because once a command has matched one stream, the other streams are not checked.

invalidTm (list)

list of streams on which invalid telemetry packets are sent. These may be used in the data links configuration, to allow saving the telemetry packets which are declared by the preprocessor as invalid (and thus not sent for further processing on the normal tm stream).

cmdHist (list)

streams used for the command history. No additional option in addition to the stream name is supported.

event

streams used for events. No additional option in addition to the stream name is supported.

param

streams used for parameters. No additional option in addition to the stream name is supported.

parameterAlarm

streams used for parameter alarms. No additional option in addition to the stream name is supported.

eventAlarm

streams used for event alarms. No additional option in addition to the stream name is supported.

sqlFile (string)

this is not a stream type but a reference to a file cotaining Stream sql statements that will be executed on instance startup. The file can create additional (non-standard) streams or tables.

4.2 Generic Archive

4.2.1 Telemetry Packets

This table is created by the *XTCE TM Recorder* (page 103) and uses the generation time and sequence number as primary key:

```
CREATE TABLE tm(
    gentime TIMESTAMP,
    seqNum INT,
    packet BINARY,
    pname ENUM,
    PRIMARY KEY(
        gentime,
        seqNum
    )
 ) HISTOGRAM(pname) PARTITION BY TIME_AND_VALUE(gentime, pname) TABLE_FORMAT=compressed;
```

Where the columns are:

- **gentime**
  generation time of the packet.

- **seqNum**
  an increasing sequence number.
- **packet**
  the binary packet.
- **pname**
  the fully-qualified name name of the container. In a container hierarchy, one has to configure which containers are used as partitions. This can be done by setting a flag in the spreadsheet.

If a packet arrives with the same time and sequence number as another packet already in the archive, it is considered duplicate and shall not be stored.

The `HISTOGRAM(pname)` clause means that Yamcs will build an overview that can be used to quickly see when data for the given packet name is available in the archive.

The `PARTITION BY TIME_AND_VALUE` clause means that data is partitioned in different RocksDB databases and column families based on the time and container name. Currently the time partitioning schema used is `YYYY/MM` which implies one RocksDB database per year,month. Inside that database there is one column family for each container that is used for partitioning.

Partitioning the data based on time, ensures that old data is frozen and not disturbed by new data coming in. Partitioning by container has benefits when retrieving data for one specific container for a time interval. If this is not desired, one can set the partitioning flag only on the root container (in fact it is automatically set) so that all packets are stored in the same partition.

4.2.2 Events

This table is created by the *Event Recorder* (page 97) and uses the generation time, source and sequence number as primary key:

```
CREATE TABLE events(
    gentime TIMESTAMP,
    source ENUM,
    seqNum INT,
    body PROTOBUF('org.yamcs.protobuf.Yamcs$Event'),
    PRIMARY KEY(
        gentime,
        source,
        seqNum
    )
) HISTOGRAM(source) partition by time(gentime) table_format=compressed;
```

Where the columns are:

- **gentime**
  the generation time of the command set by the originator.
- **source**
  a string representing the source of the events.
- **seqNum**
  a sequence number provided by the event source. Each source is expected to keep an independent sequence count for the events it generates.

4.2.3 Command History

This table is created by the *Command History Recorder* (page 97) and uses the generation time, origin and sequence number as primary key:

```
CREATE TABLE cmdhist(
    gentime TIMESTAMP,
    origin STRING,
    seqNum INT,
    cmdName STRING,
    binary BINARY,
    PRIMARY KEY(
        gentime,
        origin,
        seqNum
    )
) HISTOGRAM(cmdName) PARTITION BY TIME(gentime) table_format=compressed;
```

Where the columns are:

- **gentime**
  the generation time of the command set by the originator.
- **origin**
  a string representing the originator of the command.
- **seqNum**
  a sequence number provided by the originator. Each command originator is supposed to keep an independent sequence count for the commands it sends.
- **cmdName**
  the fully qualified name of the command.
- **binary**
  the binary packet contents.

In addition to these columns, there will be numerous dynamic columns set by the command verifiers, command releasers, etc.

Recording data into this table is setup with the following statements:

```
INSERT_APPEND INTO cmdhist SELECT * FROM cmdhist_realtime;
INSERT_APPEND INTO cmdhist SELECT * FROM cmdhist_dump;
```

The `INSERT_APPEND` clause says that if a tuple with the new key is received on one of the cmdhist_realtime or cmdhist_dump streams, it will be just inserted into the `cmdhist` table. If however, a tuple with a key that already exists in the table is received, the columns that are new in the newly received tuple are appended to the already existing columns in the table.

4.2.4 Alarms

This table is created by the *Alarm Recorder* (page 96) and uses the trigger time, parameter name and sequence number as primary key:

```
CREATE TABLE alarms(
    triggerTime TIMESTAMP,
    parameter STRING,
    seqNum INT,
    PRIMARY KEY(
        triggerTime,
        parameter,
        seqNum
    )
) table_format=compressed;
```

Where the columns are:

- **triggerTime**
  the time when the alarm has been triggered. Until an alarm is acknowledged, there will not be a new alarm generated for that parameter (even if it were to go back in limits)

- **parameter**
  the fully qualified name of the parameter for which the alarm has been triggered.
- **seqNum**
  a sequence number increasing with each new triggered alarm. The sequence number will reset to 0 at Yamcs restart.

4.2.5 Parameters

This table is created by the *Parameter Recorder* (page 101) and uses the generation time and sequence number as primary key:

```
CREATE TABLE pp(
    gentime TIMESTAMP,
    ppgroup ENUM,
    seqNum INT,
    rectime TIMESTAMP,
    primary key(
        gentime,
        seqNum
    )
) histogram(ppgroup) PARTITION BY TIME_AND_VALUE(gentime,ppgroup) table_format=compressed;
```

Where the columns are:

- **gentime**
  the generation time of the command set by the originator.
- **ppgroup**
  a string used to group parameters. The parameters sharing the same group and the same timestamp are stored together.
- **seqNum**
  a sequence number supposed to be increasing independently for each group.
- **rectime**
  the time when the parameters have been received by Yamcs.

In addition to these columns that are statically created, the pp table will store columns with the name of the parameter and the type `PROTOBUF(org.yamcs.protobuf.Pvalue$ParameterValue)`.

Note: Because `partitioning by ppgroup` is specified, this is also implicitly part of the primary key, but not stored as such in the RocksDB key.

Yamcs Generic Archive is composed of tables that store data emitted by streams.

Like streams, the tables have a variable number of columns of predefined types. Tables have a primary key composed of one or more columns. The primary key columns are mandatory, a tuple that does not have them will not be stored in the table.

The primary key is used to sort the data. Yamcs uses a (key, value) storage engine (currently RocksDB) for storing the data. Both key and value are byte arrays. Yamcs uses the serialized primary key of the table as the key in RocksDb and the remaining columns serialized as the value.

Although not enforced by Yamcs, it is usual to have the time as part of the primary key.

Yamcs stores time ordered tuples (t, v_1, $v_2 \ldots v_n$) where t is the time and v_1, v_2, v_n are values of various types. The tables are row-oriented and optimized for accessing entire records (e.g. a packet or a group of processed parameters).

Yamcs defines a standard set of tables for storing raw telemetry packets, commands, events, alarms and processed parameters.

4.3 Parameter Archive

4.3.1 Archive Filling

There are two fillers that can be used to populate Parameter Archive:

- **Realtime Filling**

  The RealtimeFillerTask will subscribe to a realtime processor and write the parameter values to the archive.

- **Backfilling**

  The ArchiveFillerTask will create from time to time replays from the raw data in the *Telemetry Packets* (page 46) and *Parameters* (page 49) tables of the Generic Archive.

Due to the fact that data is stored in segments, one segment being a value in the (key, value) RocksDB, it is not efficient to write one row (data corresponding to one timestamp) at a time. It is much more efficient to collect data and write entire or at least partial segments at a time.

The realtime filler will write the partial segments to the archive at each configurable interval. When retrieving data from the Parameter Archive, the latest (near realtime) data will be missing from the archive. That is why Yamcs uses the processor parameter cache to retrieve the near-realtime values.

The backFiller is by default enabled and it can also be used to issue rebuild requests over HTTP. The realtimeFiller has to be enabled in the configuration and the flushInterval (how often to flush the data in the archive) has to be specified. The flushInterval has to be smaller than the duration configured in the parameter cache.

The backFiller is configured with a so called warmupTime (by default 60 seconds) which means that when it performs a replay, it starts the replay earlier by the specified warmupTime amount. The reason is that if there are any algorithms that depend on some parameters in the past for computing the current value, this should give them the chance to warmup. The data generated during the warmup is not stored in the archive (because it is part of the previous segment).

4.3.2 Parameter Archive Internals

The Parameter Archive stores for each parameter tuples (t_i, ev_i, rv_i, ps_i). In Yamcs the timestamp is 8 bytes long, the raw and engineering values are of usual types (signed/unsigned 32/64 integer, 32/64 floating point, string, boolean, binary) and the parameter status is a protobuf message.

In a typical space data stream there are many parameters that do not change very often (like an device ON/OFF status). For these, the space required to store the timestamp can greatly exceed in size the space required for storing the value (if simple compression is used).

In fact since the timestamps are 8 bytes long, they equal or execed in size the parameter values almost in all cases, even for parameters that do change.

To reduce the size of the archive, some alternative parameter archives may choose to store only the values when they change with respect to the previous value. Often, like in the above “device ON/OFF” example, the exact timestamps of the non-changing parameter values, received in between actual (but rare) value changes are not very important. One has to take care that gaps in the data are not mistaken for non-changing parameter values. Storing the values on change only will reduce the space required not only for the value but also (and more importantly) for the timestamp.

However, we know that more often than not parameters are not sampled individually but in packets or frames, and many (if not all) the parameters from one packet share the same timestamp.

Usually some of the parameters in these packets are counters or other things that do change with each sampling of the value. It follows that at least for storing those ever changing parameter values, one has to store the timestamps anyway.

This is why, in Yamcs we do not adopt the “store on change only” strategy but a different one: we store the timestamps in one record and make reference to that record from all the parameters sharing those same timestamps. Of course it wouldn't make any sense to reference one singe timestamp value, instead we store multiple values in a segment and reference the time segment from all value segments that are related to it.

4.3.2.1 Archive Structure

We have established that the Yamcs Parameter Archive stores rows of data of shape: (t, pv_0, pv_1, pv_2,. . . , pv_n)

Where pv_0, pv_1, pv_2..pv_n are parameter values (for different parameters) all sharing the same timestamp t. One advantage of seeing the data this way is that we do keep together parameters extracted from the same packet (and having the same timestamp). It is sometimes useful for operators to know a specific parameter from which packet has been extracted (e.g. which APID, packet ID in a CCSDS packet structure).

The Parameter Archive partitions the data at two levels:

1. time partitioned in partitions of 2^{31} milliseconds duration (~ 25 days). Each partition is stored in its own ColumnDataFamily in RocksDB (which means separate files and the possibility to remove an entire partition at a time).
2. Inside each partition, data is segmented in segments of 2^{22} milliseconds (~ 70 minutes) duration. One data segment contains all the engineering values or raw values or parameter status for one parameter. A time segment contains all the corresponding timestamps.

This means that each parameter requires each ~70 minutes three segments for storing the raw, engineering and status plus a segment containing the timestamps. The timestamp segment is shared with other parameters. In order to be able to efficiently compress and work with the data, one segment stores data of one type only.

Each (parameter_fqn, eng_type, raw_type) combination is given an unique 4 bytes parameter_id (fqn = fully qualified name). We do this in order to be able to accomodate changes in parameter definitions in subsequent versions of the mission database.

The parameter_id=0 is reserved for the timestamp.

A ParameterGroup represents a list of parameter_id which share the same timestamp.

Each ParameterGroup is given a ParameterGroup_id

4.3.2.2 Column Families

For storing metadata we have 2 CFs:

- **meta_p2pid**

  contains the mapping between the fully-qualified parameter name and parameter_id and type

- **meta_pgid2pg**

  contains the mapping between ParameterGroup_id and parameter_id

For storing parameter values and timestamps we have 1CF per partition: data_<partition_id>. partition_id is basetimestamp (i.e. the start timestamp of the 2^{31} long partitions) in hexadecimal (without 0x in front)

Inside the data partitions we store (key, value) records where:

- **key**

parameter_id, ParameterGroup_id, type, segment_start_time (the type = 0, 1 or 2 for the eng value, raw value or parameter status)

- **value**

  ValueSegment or TimeSegment (if parameter_id = 0)

We can notice from this organization, that inside one partition, the segments containing data for one parameter follows in the RocksDB files in sequence of $\text{engvalue}_{\text{segment\_1}}$, $\text{rawvalue}_{\text{segment\_1}}$, $\text{parameterstatus}_{\text{segment\_1}}$, $\text{engvalue}_{\text{segment\_2}}$, $\text{rawvalue}_{\text{segment\_2}}, \dots$

4.3.2.3 Segment Encoding

The segments are compressed in different ways depending on their types.

- **SortedTimeSegment**

  Stores the timestamps as uitn32 deltas from the beginning of the segment. The data is first encoded into deltas of deltas, then it's zigzag encoded (such that it becomes positive) and then it's encoded with FastPFOR and VarInt. FastPFOR encodes blocks of 128 bytes so VarInt encoding is used for the remaining data.

  Storing timestamps as deltas of deltas helps if the data is sampled at regular intervals (especially by a real-time sytem). In this case the encoded deltas of deltas become very close to 0 and that compresse very well.

  Description of the VarInt and zigzag encoding can be found in Protocol Buffer docs[27].

  Description and implementation of the FastPFOR algorithm can be found at https://github.com/lemire/JavaFastPFOR.

- **IntSegment**

  Stores int32 or uint32 encoded same way as the time segment.

- **FloatSegment**

  Stores 32 bits floating point numbers encoded using the algorithm described in the Facebook Gorilla paper[28] (slightly modified to work on 32 bits).

- **ParameterStatusSegment**, **StringSegment** and **BinarySegment**

[27] https://developers.google.com/protocol-buffers/docs/encoding
[28] http://www.vldb.org/pvldb/vol8/p1816-teller.pdf

These are all stored either raw, as an enumeration, or run-length encoded, depending on which results in smaller compressed size.

- **DoubleSegment** and **LongSegment**

  These are only stored as raw for the moment - compression remains to be implemented. For DoubleSegment we can employ the same approach like for 32 bits (since the original approach is in fact designed for compressing 64 bits floating point numbers).

4.3.2.4 Future Work

- **Segment Compression**

  Compression for DoubleSegment and LongSegment. DoubleSegment is straightforward, for LongSegment one has to dig into the FastPFOR algorithm to understand how to change it for 64 bits.

- **Archive Filling**

  It would be desirable to backfill only parts of the archive. Indeed, some ground generated data may not suffer necessarily of gaps and could be just realtime filled. Currently there is no possibility to specify what parts of the archive to be back-filled.

  Another useful feature would be to trigger the back filling automatically when gaps are filled in Yamcs database tables.

The Parameter Archive stores time ordered parameter values. The parameter archive is column oriented and is optimized for accessing a (relatively small) number of parameters over longer periods of time.

The Parameter Archive stores for each parameter tuples of (t_i, ev_i, rv_i, ps_i) where:

t_i the *generation* timestamp of the value. The *reception* timestamp is not stored in the Parameter Archive.

ev_i the engineering value of the parameter at the given time.

rv_i the engineering value of the parameter at the given time.

ps_i the parameter status of the parameter at the given time.

The parameter status includes attributes such as out-of-limits indicators (alarms) and processing status. Yamcs Mission Database provides a mechanism through which a parameter can change its alarm ranges depending on the context. For this reason the Parameter Archive also stores the parameter status and the applicable alarm ranges at the given time.

In order to speed up the retrieval, the Parameter Archive stores data in segments of approximately 70 minutes. That means that all engineering values for one parameter for the 70 minutes are stored together; same for raw values, parameter status and timestamps.

Having all the data inside one segment of the same type offers possibility for good compression especially if the values do not change much or at all (as it is often the case).

While this structure is good for fast retrieval, it does not allow updating data very efficiently and in any case not in realtime. This is why the Parameter Archive is filled in batch mode. Data is accumulated in memory and flushed to disk periodically using different filling strategies.

4.4 Object Archive (buckets)

The Yamcs object archive is used to store general data objects (images, files, etc) which are generally unstructured information. The objects are grouped into `buckets`; each bucket has a name and is simply a collection of related objects.

Inside a bucket each object is identified by an name and has associated a set of metadata. The name is usually (but not necessarily) a UNIX directory like path "/a/b/c/" and the metadata is a list of `key: value` where both the key and the value are strings.

Yamcs supports two ways of storing the objects: inside the RocksDB database or on the server filesystem as files. For RocksDB buckets, each object is stored in a (key, value) record, the key is the object name prepended by a prefix identifying the bucket and the value is the object data. For filesystems buckets, each bucket represents a directory on disk and the objects are the files in that directory (including subdirectories). The filesystem buckets do not support metadata currently.

A bucket is limited to 100MB in size and maximum 1000 objects. In addition, the HTTP API imposes a limit of 5MB for each uploaded object. Note that since the filesystem buckets can be changed outside Yamcs (just copying files in a directory) the total size limit or the number of objects limit may be exceeded.

The RocksDB buckets can be created in the configuration or programmatically using the HTTP API[29].

The filesystem buckets can only be defined in the configuration by using the `buckets` configuration option as shown below.

The `buckets` keyword in yamcs.yaml will define a list of buckets.

```yaml
buckets:
  - name: cfdpUp
    path: ../../cfdpUp
```

4.4.1 Options

name (string)
The name of the bucket. The name must contain only letters, digits or _.

path (string)
If this option is present the bucket is a filesystem bucket and a directory with the given path will be created if not already existing. If omitted, this bucket will be stored binary in the Yamcs database (RocksDB).

maxSize (number)
The maximum allowed size of the bucket in bytes.

maxObjects (number)
The maximum allowed number of objects in this bucket.

Note: The *maxSize* and *maxObjects* are enforced when *new* objects are added to the bucket. It is possible for limits to be lower than the actual usage. For example, when they have been reconfigured. Or, in the case of filesystem buckets, because content has changed outside of Yamcs.

[29] https://docs.yamcs.org/yamcs-http-api/buckets

5. Data Links

Data Links represent special components that communicate with the target instrument or spacecraft. There are three types of Data Links: TM, TC and PP (processed parameters). TM and PP receive telemetry packets or parameters and inject them into the realtime or dump TM or PP streams. The TC data links subscribe to the realtime TC stream and send data to the external systems.

Data Links can report on their status and can also be controlled by an operator to connect or disconnect from their data source.

Note that any Yamcs Service can connect to external sources and inject data in the streams. Data links however, can report on their status using a predefined interface and can also be controlled to connect or disconnect from their data source.

Data links are defined in `etc/yamcs.(instance).yaml`. Example:

```
dataLinks:
  - name: tm_realtime
    class: org.yamcs.tctm.TcpTmDataLink
    enabledAtStartup: true
    stream: tm_realtime
    invalidPackets: DIVERT
    invalidPacketsStream: invalid_tm_stream
    ....
```

General configuration options.

name (string)

Required. The name that will be assigned to the link. Each link needs a unique name; the name can be seen in the user interface and can be used for API calls.

class (string)

Required. The name of the class that is implementing the link. The class has to implement the Link[30] interface.

enabledAtStartup (boolean)

If set to false, the link will be disabled at startup. By default it is true. The link can be enabled/disabled at any time via an API call.

stream (string)

The name of the stream where the data is taken from or injected into.

tmStream (string)

This is an alternative to *stream*; can be used for links serving more than one of TM, TC or PP (processed parameters).

tcStream (string)

This is an alternative to *stream*; can be used for links serving more than one of TM, TC or PP.

ppStream (string)

This is an alternative to *stream*; can be used for links serving more than one of TM, TC or PP.

[30] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/Link.html

invalidPackets (string)

One of `DROP`, `PROCESS` or `DIVERT`. Used for TM links to specify what happens with the packets that the pre-processor decides are invalid:

`DROP` means they are discarded.

`PROCESS` means they are put on the normal stream (configured with the `stream` parameter), same like the valid packets.

`DIVERT` means they are put on another stream specified by the option `invalidPacketsStream`.

invalidPacketsStream (string)

If `invalidPackets` is set to `DIVERT`, this configures the stream where the packets are sent.

updateSimulationTime (boolean)

If set to true, the link will update the simulation time using the generation time of each packet received. The SimulationTimeService has to be configured for the instance, otherwise an error will be raised at startup.

Other options are link-specific and documented in their respective sections.

5.1 Packet Pre-processor

Yamcs generally uses the Mission Database to process telemetry packets. When data is received from external systems, there are two processing steps done as part of the Data Link which are outside the Mission Database definition:

1. Splitting a data stream into packets. This is done only for the links that receive data as a stream (e.g. TCP). For Data Links where input is naturally split into frames (e.g. UDP) this step is not necessary and not performed.
2. Pre-processing of packets in order to detect/correct errors and to retrieve basic information about the packets.

5.1.1 Stream Splitting

The data stream splitter is a java class that implements the PacketInputStream[31] interface.

A generic splitter for binary streams is defined in GenericPacketInputStream[32]. This class can split a stream based on a packet length that is encoded in a header. It requires all packets to have the length on the same number of bytes.

5.1.2 Packet pre-processing

The packet pre-processor is a java class that implements the PacketPreprocessor[33] interface.

It is responsible for error detection (and possibly correction) and extracting basic information required for further packet processing:

- packet generation time - it represents the time when the packet has been generated on-board.
- sequence count - a number used to distinguish two packets having the same timestamp.

The generation time and sequence count are used as primary key in the tm table in the archive. That means they have to uniquely identify a packet; if the archive receives a new packet with the same (generation time, sequence count) as an existing packet in the archive, it will be considered a duplicate and discarded.

[31] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/PacketInputStream.html

[32] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/GenericPacketInputStream.html

[33] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/PacketPreprocessor.html

The sequence count is used to distinguish two packets that have the same timestamp; it does not need to be incremental. For example the ISSPacketPreprocessor[34] uses the first 4 bytes of the CCSDS primary header (containing APID and CCSDS sequence count among others) as sequence count for the telemetry stream.

Each mission has specific ways to encode information in the header but there are some standards supported to a certain extent by Yamcs:

- Packet Utilisation Standard (PUS) from ESA - implemented in PusPacketPreprocessor[35].
- NASA cFS - implemented in CfsPacketPreprocessor[36].

Generation time

A particular difficulty when writing a pre-processor is dealing with the generation time. Yamcs originated in the ISS (International Space Station) world where all the payloads and instruments are time synchronized to GPS and each packet sent to ground has a reliable timestamp. This is of course not true for all spacecrafts - most on-board computer have just an internal clock count which resets to 0 when the computer is restarted.

The Yamcs archive needs the generation time for all its functions, not having it means that a large part of the functionality of Yamcs is not usable.

There are different mechanisms to synchronize the on-board time with the ground:

- Do not attempt to synchronize the time. The pre-processor can use local generation (computer) reception time as generation time. The on-board time will be still available as a parameter if defined in the MDB. This method is especially useful when using Yamcs as part of a test and check-out system, the system under test might be incomplete and have no (reliable) clock at all. The disadvantage is that when receiving data in non-realtime (e.g. recorded on board or in a ground station), it will not fit orderly in the archive.
- Synchronize the on-board system to the ground each time it resets. This is the method employed by cFS - it allows setting a spacecraft time correction factor (STCF) on-board and that will make the on-board time correlated to the ground.
- Maintain a correlation factor on ground, his is the method specified by ESA PUS standard. In this case the packet pre-processor has to implement the time correlation. The *Time Correlation Service* (page 104) can be used to correlate the on-board time with the ground time.

Regardless of which method is used, it is important that the pre-processor does not generate packets with wrong timestamps - these might be difficult to locate and remove from the archive later.

Time Decoding

The packet pre-processors can use time decoders to decode the time from the packet. The time decoders are classes implementing the TimeDecoder[37] interface. All the pre-processors extending the AbstractPacketPreprocessor[38] will have access to the time decoders configured by the `timeEncoding` option.

The time decoders are responsible for providing a relative time in milliseconds; the relative time is converted to an absolute time using a specified epoch.

If there is no epoch specified, the time is considered `raw` and the *Time Correlation Service* (page 104) service is used for converting the time to an absolute time. This is the case when the on-board time is not synchronized to anything and the time in the packet is the value of an on-board computer clock which is just a counter most likely initialized at 0 when the on-board computer resets. The raw times do not have units, it is up to the time decoder to decide what value to return; the requirement however is to be linearly correlated to the time. The time correlation service will compute the gradient and the offset that can be used to convert the raw value to an absolute time.

There are a few common options for all time decoders:

epoch (string)

[34] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/ISSPacketPreprocessor.html
[35] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/pus/PusPacketPreprocessor.html
[36] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/cfs/CfsPacketPreprocessor.html
[37] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/time/TimeDecoder.html
[38] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/AbstractPacketPreprocessor.html

Specifies to which epoch the time relates to. Can be one of:

- TAI - the time is a delta from 1-Jan-1958, as recomended by CCSDS Time Code Formats.
- J2000 - the time is a delta from J2000 epoch which corresponds to 2000-01-01T11:58:55.816 UTC.
- GPS - the time is a delta from GPS epoch which corresponds to 1980-01-6T00:00:00 UTC.
- UNIX - the time corresponds to the time as kept by UNIX - that is a pseudo-number of seconds from 1-Jan-1970. We say "pseudo" because this time does not include leap seconds and therefore it is not a true delta time from the epoch (and the epoch is anyway not well defined). However that number can be used to calculate a UTC time (by appliying Gregorian-calendar conventions). Yamcs will convert that time to the internal time format by adding the leap seconds.
- CUSTOM - the time corresponds to a delta or pseudo delta specified in the option `epochUTC`.
- NONE - the time read from the packet is not a delta from an epoch but rather the value of free running clock . A time correlation service can be used to translate that value to a real time.

epochUTC (iso8601 string)
If the epoch is defined as `CUSTOM`, can be used to specify the UTC time from which the decoded time is a delta or pseudo-delta.

timeIncludesLeapSeconds: (boolean)
If the epoch is defined as `CUSTOM`, can be used to specify if the time read from that epoch includes the leap seconds (meaning it is a true delta time). If the value is false, Yamcs will add the missing leap seconds between the time specified in the epochUTC and the time read from the packet. From the 4 standard epochs (TAI, J2000, GPS and UNIX), only the UNIX time will have this set to false. Default: true

Two time decoder types are currently implemented: CUC and FIXED.

CUC time decoder

`CUC` which is an abbreviation for CCSDS Unsegmented time Code. *Unsegmented* means that the entire time field can be seen as a continuous integer counter of the fractional time unit. A segmented time code for example one which provides days and millisecond of the day and in which a 32 bit field is used to represent the millisecond of the day is not continuous because there are less than 2^32 milliseconds in a day.

The time is decoded as specified in CCSDS Time Code Formats CCSDS 301.0-B-4[39], Chapter 3.2. In short the time is encoded as an optional 1 or 2 bytes `pfield` (preamble field) followed by a 1-7 bytes basic time followed by a 0-10 bytes fractional time. The `pfield` specifies the length in bytes of the basic and fractional times.

For example `pfield = 0x2E` means that the basic time is encoded on 4 bytes and the fractional time is encoded on 2 bytes, making the length of the time in the packet 6 bytes when the `pfield` is implicit or 7 bytes when it is part of the packet.

The `pfield` contains some information about the epoch used. This information is ignored, the epoch is configured with the `epoch` option, as described below.

The standard allows in principle more than 2 `pfield` bytes but this is not supported (a custom time decoder has to be used in this case).

The CUC decoder can work in two modes depending whether the time decoded is a delta time from a configured epoch or the value of a free running on-board clock.

[39] https://public.ccsds.org/Pubs/301x0b4e1.pdf

If the time decoded is a delta time from a configured epoch (`epoch` is different than `NONE`), the CUC decoder assumes the basic time unit to be the second and it decodes the time to a delta or pseudo-delta from the epoch. The precision is milliseconds (as all time storage in Yamcs), irrespective of the precision used in the encoded time - this means that at maximum two bytes of fractional time will be used. If the fractional time is 2 bytes (i.e. each fractional unit is 1/2^16 seconds) or more, it will be be down-rounded when converted to Yamcs time. The maximum length of supported basic time is 6 bytes; this is because 7 or more bytes cannot be converted to 64 bits milliseconds.

When the decoded time is the value of a free running on-board clock (epoch is `NONE`), the CUC decoder provides the "raw" time in the unit of the fractional time (without any precision loss). The time is decoded as a big endian value on bn+fn bytes where bt is the number of basic time bytes and fn is the number of fractional time bytes (as read from the `pfield`). Practically in this case the decoder doesn't make distinction between basic time and fractional time (this works because the time is unsegmented). The value thus obtained is expected to be passed to a *Time Correlation Service* (page 104) which will convert it to an actual time, automatically detecting the unit of the fractional time.

The maximum supported length of the "raw" time is 8 bytes, if the time is encoded on 9 or more bytes, an exception will be thrown in the decodeRaw() method.

CUC decoder configuration options:

type (string)
Has to be `CUC` to select the CUC decoder.

implicitPField (integer)
If the `pfield` is not encoded in the packet, it can be set by this option.

A value of -1 means that the `pfield` is explicitly provided in the packet. Default: -1.

implicitPFieldCont (integer)
This can be used to configure the next octet of the `pfield` in case the first bit of the first octet (specified above) is 1.

FIXED time decoder

The FIXED decoder decodes the time as a signed integer on 4 or 8 bytes and has an optional multiplier to convert the integer to milliseconds. The multiplier is not used when decoding the time as raw time (i.e. when the epoch is NONE).

FIXED decoder options:

type (string)
Has to be `FIXED` to select the FIXED decoder.

size(integer)
number of bytes containing the time. It has to be 4 or 8. Default: 8

multiplier (double)
used to transform the extracted integer to milliseconds. Default: 1.0

5.1.3 Pre-processor Configuration

The AbstractPacketPreprocessor[40] provides some general configuration options which can be used in custom pre-processors and are used in the PUS and cFS pre-processors.

Example

```
dataLinks:
  - name: tm_realtime
    ...
    packetPreprocessorClassName: org.yamcs.tctm.pus.PusPacketPreprocessor
```

(continues on next page)

[40] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/AbstractPacketPreprocessor.html

(continued from previous page)

```
    packetPreprocessorArgs:
      errorDetection:
         type: CRC-16-CCIIT
      useLocalGenerationTime: false
      timeEncoding:
         type: CUC
         epoch: CUSTOM
         epochUTC: "2010-09-01T00:00:00Z"
         timeIncludesLeapSeconds: true
      tcoService: tco0
```

Configuration Options

errorDetection (map)

If specified, the *errorDetectionCalculator* object will be made available to the pre-processor to calculate the CRC used to verify the integrity of the packet. The sub-options are:

type (string)

Required Can take one of the values:

- 16-SUM - calculates a 16 bits checksum over the entire packet which has to contain an even number of bytes. This checksum is used in Columbus/ISS data.
- CRC-16-CCIIT - standard CRC algorithm used in PUS and also in CCSDS standards for frame encoding.
- ISO-16 - specified in PUS as alternative to CRC-16-CCIIT.
- NONE - no error detection will be used, this is the default if the errorDetection map is not present.

initialValue (integer)

Used when the type is `CRC-16-CCIIT` to specify the initial value used for the algorithm. Default: `0xFFFF`.

userLocalGenerationTime (boolean)

If true, the packets will be timestamp with local mission time rather than the time extracted from the packets. Default: false.

timeEncoding (map)

This contains instructions from how to read the time from the packet. See above for description on how to configure the time decoder.

5.2 Command Post-Processor

Similar to the TM packet pre-processors, the command post-processors are used to change the command before being sent out on the data link. The post-processors are java classes that implement the Command-Postprocessor[41] interface.

Typical tasks performed by the post-processors are:

- assigning a sequence count (e.g. the CCSDS sequence counts are assigned per APID)
- computing and appending a checksum or CRC

[41] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/CommandPostprocessor.html

5.3 File Polling TM Data Link

Reads data from files in a directory, importing it into the configured stream. The directory is polled regularly for new files and the files are imported one by one. After the import, the file is removed.

5.3.1 Class Name

org.yamcs.tctm.FilePollingTmDataLink[42]

5.3.2 Configuration Options

stream (string)
Required. The stream where data is emitted

incomingDir (string)
The directory where the data will be read from. If not specified, the data will be read from `<yamcs-incoming-dir>/<instance>/tm/` where `yamcs-incoming-dir` is the value of the incomingDir property in `etc/yamcs.yaml`.

deleteAfterImport (boolean)
Remove the file after importing all the data. By default set to true, can be set to false to import the same data again and again.

delayBetweenPackets (integer)
When importing a file, wait this many milliseconds after each packet. This option together with the previous one can be used to simulate incoming realtime data.

packetPreprocessorClassName (string)
Class name of a PacketPreprocessor[43] implementation. Default is org.yamcs.tctm.IssPacketPreprocessor[44] which applies ISS conventions.

packetPreprocessorArgs (map)
Optional args of arbitrary complexity to pass to the PacketPreprocessor. Each PacketPreprocessor may support different options.

5.4 TCP TC Data Link

Sends telecommands via TCP.

5.4.1 Class Name

org.yamcs.tctm.TcpTcDataLink[45]

5.4.2 Configuration Options

stream (string)
Required. The stream where command instructions are received

host (string)
Required. The host of the TC provider

[42] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/FilePollingTmDataLink.html
[43] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/PacketPreprocessor.html
[44] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/IssPacketPreprocessor.html
[45] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/TcpTcDataLink.html

port (integer)
Required. The TCP port to connect to

tcQueueSize (integer)
Limit the size of the queue. Default: unlimited

tcMaxRate (integer)
Ensure that on overage no more than `tcMaxRate` commands are issued during any given second. Default: unspecified

commandPostprocessorClassName (string)
Class name of a CommandPostprocessor[46] implementation. Default is org.yamcs.tctm.IssCommandPostprocessor[47] which applies ISS conventions.

commandPostprocessorArgs (map)
Optional args of arbitrary complexity to pass to the CommandPostprocessor. Each CommandPostprocessor may support different options.

5.5 TCP TM Data Link

Provides packets received via plain TCP sockets.

In case the TCP connection with the telemetry server cannot be opened or is broken, it retries to connect each 10 seconds.

5.5.1 Class Name

org.yamcs.tctm.TcpTmDataLink[48]

5.5.2 Configuration Options

host (string)
Required. The host of the TM provider

port (integer)
Required. The TCP port to connect to

stream (string)
Required. The stream where data is emitted

packetInputStreamClassName (string)
Class name of a PacketInputStream[49]. Default is org.yamcs.tctm.CcsdsPacketInputStream[50] which reads CCSDS Packets.

packetInputStreamArgs (map)
Optional args of arbitrary complexity to pass to the PacketInputStream. Each PacketInputStream may support different options.

packetPreprocessorClassName (string)
Class name of a PacketPreprocessor[51] implementation. Default is org.yamcs.tctm.IssPacketPreprocessor[52] which applies ISS conventions.

[46] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/CommandPostprocessor.html
[47] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/IssCommandPostprocessor.html
[48] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/TcpTmDataLink.html
[49] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/PacketInputStream.html
[50] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/CcsdsPacketInputStream.html
[51] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/PacketPreprocessor.html
[52] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/IssPacketPreprocessor.html

packetPreprocessorArgs (map)
Optional args of arbitrary complexity to pass to the PacketPreprocessor. Each PacketPreprocessor may support different options.

5.6 TSE Data Link

Sends telecommands to a configured *../services/global/tse-commander* and reads back output as processed parameters.

5.6.1 Class Name

org.yamcs.tctm.TseDataLink[53]

5.6.2 Configuration Options

host (string)
Required. The host of the TSE Commander.

port (integer)
Required. The TCP port of the TSE Commander.

tcStream (string)
Stream where command instructions are received. Default: `tc_tse`.

ppStream (string)
Stream where to emit received parameters. Default: `pp_tse`.

5.7 UDP Parameter Data Link

Listens on a UDP port for datagrams containing Protobuf encoded messages. One datagram is equivalent to a message of type ParameterData[54]

5.7.1 Class Name

org.yamcs.tctm.UdpParameterDataLink[55]

5.7.2 Configuration Options

stream (string)
Required. The stream where data is emitted

port (integer)
Required. The UDP port to listen on

5.8 UDP TC Data Link

Sends telecommands via UDP socket. One datagram is equivalent to one command.

[53] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/TseDataLink.html
[54] https://yamcs.org/javadoc/yamcs/org/yamcs/protobuf/Pvalue/ParameterData.html
[55] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/UdpParameterDataLink.html

5.8.1 Class Name

org.yamcs.tctm.UdpTcDataLink[56]

5.8.2 Configuration Options

stream (string)
Required. The stream where data is emitted

host (string)
Required. The host of the TC provider

port (integer)
Required. The UDP port to send to

port (integer)
Required. The UDP port to listen on

tcQueueSize (integer)
Limit the size of the queue. Default: unlimited

tcMaxRate (integer)
Ensure that on overage no more than `tcMaxRate` commands are issued during any given second. Default: unspecified

commandPostprocessorClassName (string)
Class name of a CommandPostprocessor[57] implementation. Default is org.yamcs.tctm.IssCommandPostprocessor[58] which applies ISS conventions.

commandPostprocessorArgs (map)
Optional args of arbitrary complexity to pass to the CommandPostprocessor. Each CommandPostprocessor may support different options.

5.9 UDP TM Data Link

Listens on a UDP port for datagrams containing CCSDS packets. One datagram is equivalent to one packet.

5.9.1 Class Name

org.yamcs.tctm.UdpTmDataLink[59]

5.9.2 Configuration Options

stream (string)
Required. The stream where data is emitted

port (integer)
Required. The UDP port to listen on

maxLength (integer)
The maximum length of the packets received. If a larger datagram is received, the data will be truncated. Default: 1500 bytes

[56] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/UdpTcDataLink.html
[57] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/CommandPostprocessor.html
[58] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/IssCommandPostprocessor.html
[59] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/UdpTmDataLink.html

packetPreprocessorClassName (string)

Class name of a PacketPreprocessor[60] implementation. Default is org.yamcs.tctm.IssPacketPreprocessor[61] which applies ISS conventions.

packetPreprocessorArgs (map)

Optional args of arbitrary complexity to pass to the PacketPreprocessor. Each PacketPreprocessor may support different options.

5.10 CCSDS Frame Processing

This section describes Yamcs support for parts of the following CCSDS specifications:

- TM Space Data Link Protocol CCSDS 132.0-B-3[62]
- AOS Space Data Link Protocol CCSDS 732.0-B-4[63]
- TC Space Data Link Protocol CCSDS 232.0-B-4[64]
- Unified Space Data Link Protocol CCSDS 732.1-B-2[65]
- TC Synchronization and Channel Coding CCSDS 231.0-B-4[66]
- TM Synchronization and Channel Coding CCSDS 131.0-B-3[67]
- Communications Operation Procedure (COP-1) CCSDS 232.1-B-2[68]
- Space Packet Protocol CCSDS 133.0-B-2[69]
- Encapsulation Service CCSDS 133.1-B-3[70]

These specifications are dealing with multiplexing and to a certain extent encoding data for transmission on a space link.

The document Space Data Link Protocols — Summary of Concept and Rationale[71] provides a comprehensive summary of the different protocols and it is recommended to read it before attempting to configure Yamcs to use these protocols.

5.10.1 Telemetry Frame Processing

The CCSDS specifies how to transport data into three types of frames:

- AOS
- TM
- USLP

Yamcs supports to a certain extent all three of them. The main support is around the "packet service" - that is describing how the telemetry packets are extracted from the frames. The implementation is however generic enough (hopefully) such that it is possible to add additional functionality for processing non-packet data (e.g. sending video to external application).

[60] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/PacketPreprocessor.html
[61] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/IssPacketPreprocessor.html
[62] https://public.ccsds.org/Pubs/132x0b3.pdf
[63] https://public.ccsds.org/Pubs/732x0b4.pdf
[64] https://public.ccsds.org/Pubs/232x0b4.pdf
[65] https://public.ccsds.org/Pubs/732x1b2.pdf
[66] https://public.ccsds.org/Pubs/231x0b4e0.pdf
[67] https://public.ccsds.org/Pubs/131x0b3e1.pdf
[68] https://public.ccsds.org/Pubs/232x1b2e2c1.pdf
[69] https://public.ccsds.org/Pubs/133x0b2e1.pdf
[70] https://public.ccsds.org/Pubs/133x1b3e1.pdf
[71] https://public.ccsds.org/Pubs/130x2g3.pdf

The packets are inserted into frames which are sent as part of Virtual Channels (VC). The VCs can have different priority on-board, for example one VC can be used to transport low volume HK data, while another one to transport high volume science data.

Note that The USLP frames (as well as the TC frames used for commanding) support a second level of multiplexing called Multiplexer Access Point (MAP) which allows multiplexing data inside a VC. The MAP service is not supported by Yamcs.

Currently the built-in way to receive frame data inside Yamcs is by using the UdpTmFrameLink data link. The yamcs-sle project provides an implementation of the Space Link Extension (SLE) which allows receiving frame data from SLE-enabled Ground Stations (such as those from NASA Deep Space Network or ESA ESTRACK). The options described below are valid for both link types.

An example of a UDP TM frame link specification is below:

```yaml
- name: UDP_FRAME_IN
  class: org.yamcs.tctm.ccsds.UdpTmFrameLink
  args:
    port: 10017
    rawFrameDecoder:
        codec: RS
        interleavingDepth: 5
        errorCorrectionCapability: 16
        derandomize: false
    frameType: "AOS"
    spacecraftId: 0xAB
    frameLength: 512
    frameHeaderErrorControlPresent: true
    insertZoneLength: 0
    errorDetection: CRC16
    clcwStream: clcw
    goodFrameStream: good_frames
    badFrameStream: bad_frames
    virtualChannels:
      - vcId: 0
        ocfPresent: true
        service: "PACKET"
        maxPacketLength: 2048
        packetPreprocessorClassName: org.yamcs.tctm.IssPacketPreprocessor
        packetPreprocessorArgs:
          [...]
        stream: "tm_realtime"
      - vcId: 1
        ocfPresent: true
        service: "PACKET"
        maxPacketLength: 2048
        stripEncapsulationHeader: true
        packetPreprocessorClassName: org.yamcs.tctm.GenericPacketPreprocessor
        packetPreprocessorArgs:
          [...]
        stream: "tm2_realtime"
      - vcId: 2
        ocfPresent: true
        service: "PACKET"
        maxPacketLength: 2048
        packetPreprocessorClassName: org.yamcs.tctm.IssPacketPreprocessor
        stream: "tm_dump"
```

The following general options are supported:

rawFrameDecoder (map) supported since Yamcs 5.5.7

Decodes raw frame data using an error correction scheme and/or randomization. For the moment only the Reed-Solomon codec is supported. If this is not set, the frames are considered already decoded. See below for the options to the Reed-Solomon codec.

frameType (string)

Required. One of `AOS`, `TM` or `USLP`. The first 2 bits for AOS/TM and 4 bits for USLP represent the version number and have to have the value 0, 1 or 12 respectively. If a frame is received that has a different version, it is discarded (with a warning log message).

derandomize (boolean)
If true, derandomize the frames with the derandomizer as per CCSDS 131.0-B-3. Default: false

spacecraftId (integer)
Required. The expected spacecraft identifier. The spacecraftId is encoded in the frame header. If a frame with a different identifier is received, it is discarded (with a warning log message).

frameLength (integer)
The expected frame length. This parameter is mandatory for AOS and TM frames and optional for USLP frames which can have variable length. If a frame is received that does not have this length, it is discarded (with a warning log message). For USLP frames, if this parameter is specified, the following two are ignored; Yamcs will use maxFrameLength = minFrameLength = frameLength.

maxFrameLength (integer)
Used for USLP with variable frame length to specify the maximum length of the frame. This parameter is ignored if the frameLength parameter is also specified.

minFrameLength (integer)
Used for USLP with variable frame length to specify the minimum length of the frame. This parameter is ignored if the frameLength parameter is also specified.

frameHeaderErrorControlPresent (boolean)
Used only for AOS frames to specify the presence/absence of the 2 bytes Frame Header Error Control. This can be used to detect and correct errors in parts of the AOS frame headers using a Reed-Solomon (10,6) code.

insertZoneLength (integer)
The AOS and USLP frames can optionally use an Insert Service to transfer fixed-length data synchronized with the release of the frames. The insert data follows immediately the frame primary header. If the Insert Service is used, this parameter specifies the length of the insert data. If not used, please set it to 0 (default). For TM frames this parameter is ignored. Currently Yamcs ignores any data in the insert zone.

errorDetection (string)
One of `NONE`, `CRC16` or `CRC32`. Specifies the error detection scheme used. TM and AOS frames support either NONE or CRC16 while USLP supports NONE, CRC16 or CRC32. If present, the last 2 resepectively 4 bytes of the frame will contain an error control field. If the CRC does not match the computation, the frame will be discarded (with a warning message).

clcwStream (string)
Can be used to specify the name of the stream where the Command Link Control Words (CLCW) will be sent. The CLCW is the mechanism used by COP-1 to acknolwedge uplinked frames. For TM and USLP frames, there is an OCF flag part of the frame header indicating the presence or not of the CLCW. For AOS frames it has to be configured with the `ocfPresent` flag below. If present, the CLCW is also extracted from idle frames (i.e. frames that are inserted when no data needs to be transmitted in order to keep the constant bitrate required for downlink).

goodFrameStream (string)
If specified, the good frames will be sent on a stream with that name. The stream will be created if it does not exist.

badFrameStream (string)
If specified, the bad frames will be sent on a stream with that name. Bad frames are conisdered as those that fail decoding for various reasons: length in the header does not match the size of the data received, frame version does not match, bad CRC, bad spacecraft id, bad vcid.

virtualChannels (map)
Required. Used to specify the Virtual Channel specific configuration.

For each Virtual Channel in the `virtualChannels` map, the following parameters can be used:

vcId (integer)
Required. The configured Virtual Channel identifier.

ocfPresent: (boolean)

Used for AOS frames to indicate that the Virtual Channel uses the Operational Control Field (OCF) Service to transport the CLCW containing acknowledgemnts for the uplinked TC frames. For TM and USLP frames, there is a flag in each frame that indicates the presence or absence of OCF.

service:

Required. This specifies the type of data that is part of the Virtual Channel. One of `PACKET`, `IDLE` or `VCA`

PACKET:

This is used if the data contains packets - it requires the presence of the first header pointer to indicate where in the frame the packet starts. Both CCSDS space packets and CCSDS encapsulation packets are supported (even multiplexed on the same virtual channel). The type of packet is detected based on the first 3 bits of data: 000=CCSDS space packet, 111=encapsulation packets. Idle CCSDS space packets (having APID = 0x7FF) and idle encapsulation packets (having first byte = 0x1C) are discarded.

IDLE:

Supported for AOS and USLP to indicate that the Virtual Channel contains only idle frames . Normally, the AOS and USLP use the Virtual Channel 63 to transmit idle frames and you do not need to define this virtual channel (in conclusion `IDLE` is not very useful). The TM frames have a different mechanism to signal idle frames (first header pointer is 0x7FE).

VCA:

VCA stands for Virtual Channel Access - it is a mechanism for the user to plug a custom handler for the virtual channel data. The `vcaHandlerClassName` property has to be defined if this option is specified (see below).

maxPacketLength:

Required if service=PACKET. Specifies the maximum size of a packet (header included). Valid for both CCSDS Space Packets and CCSDS encapsulation packets. If the header of a packet indicates a packet size larger than this value, a warning event is raised and the packet is droped including all the data until a new frame containing a packet start.

packetPreprocessorClassName and packetPreprocessorArgs

Required if service=PACKET. Specfies the packet pre-processor and its configuration that will be used for the packets extracted from this Virtual Channel. See *Packet Pre-processor* (page 56) for details.

vcaHandlerClassName:

Required if the service = VCA Specifies the name of the class which handles data for this virtual channel. The class has to implement VcDownlinkHandler[72] interface. Optionally it can implement Link[73] interface to appear as a data link (e.g. in yamcs-web). An example implementation of such class can be found in the ccsds-frames example project.

Raw Frame Decoder

The options which can be selected under the `rawFrameDecoder` key are the following:

codec (string)

Required. Specifies the error correction codec to use. Valid values are `NONE` and `RS`. None means the data will not be error corrected (can be still useful if only de-randomization is required). RS means the Reed-Solomon codec is used and the errorCorrectionCapability and interleavingDepth below can be used to configure the codec.

interleavingDepth (int)

The interleving depth specifies the number of RS decoders running in “parallel” for one frame. Each interleavingDepth'th byte in the frame will be passed to a different decoder. Note howerver that as of Yamcs 5.5.7, the data is process sequentially not in parallel. Default: 5

errorCorrectionCapability (int)

This is either 8 or 16 determining the RS(255, 239) respectively RS(255,223) codec to be used. Default: 16

[72] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/ccsds/VcDownlinkHandler.html

[73] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/Link.html

derandomize (boolean)

If true, the data will be passed through a derandomizer after being decoded. Default: false

5.10.2 Telecommand Frame Processing

Yamcs supports packing telecommand packets into TC Transfer Frames and in addition encapsulating the frames into Communications Link Transmission Unit (CLTU).

Currently the built-in way to send telecommand frames from Yamcs is by using the UdpTcFrameLink data link. The yamcs-sle project provides an implementation of the Space Link Extension (SLE) which allows sending CLTUs to SLE-enabled Ground Stations. The options described below are valid for both link types.

An example of a UDP TC frame link specification is below:

```
- name: UDP_FRAME_OUT
  class: org.yamcs.tctm.ccsds.UdpTcFrameLink
  host: localhost
  port: 10018
  spacecraftId: 0xAB
  maxFrameLength: 1024
  cltuEncoding: BCH
  priorityScheme: FIFO
  randomizeCltu: false
  virtualChannels:
      - vcId: 0
        service: "PACKET"
        priority: 1
        commandPostprocessorClassName: org.yamcs.tctm.IssCommandPostprocessor
        commandPostprocessorArgs:
          [...]
        stream: "tc_sim"
        useCop1: true
        clcwStream: "clcw"
        initialClcwWait: 3600
        cop1T1: 3
        cop1TxLimit: 3
        bdAbsolutePriority: false
```

The following general options are supported:

spacecraftId (integer)

Required. The spacecraftId is encoded in the TC Transfer Frame primary header.

maxFrameLength (integer)

Required. The maximum length of the frames sent over this link. The Virtual Channel can also specify an option for this but the VC specifc maximum frame length has to be smaller or equal than this. Note that since Yamcs does not support segmentation (i.e. spliting a TC packet over multiple frames), this value limits effectively the size of the TC packet that can be sent.

priorityScheme (string)

One of `FIFO`, `ABSOLUTE` or `POLLING_VECTOR`. This configures the priority of the different Virtual Channels. The different schemes are described below.

cltuEncoding (string)

One of `BCH`, `LDPC64`, `LDPC256`, or `CUSTOM`. If this parameter is present, the TC transfer frames will be encoded into CLTUs and this parameter configures the code to be used. If this parameter is not present, the frames will not be encapsulated into CLTUs and the following related parameters are ignored. If the value is `CUSTOM`, the CLTU generator class must be specified as indicated below.

cltuStartSequence (string)

This parameter can optionally set the CLTU start sequence in hexadecimal if different than the CCSDS specs.

cltuTailSequence (string)

This parameter can optionally set the CLTU tail sequence in hexadecimal if different than the CCSDS specs.

randomizeCltu (boolean)
Used if cltuEncoding is BCH or CUSTOM to enable/disable the randomization. For LDPC encoding, randomization is always on. Note that as per issue 4 of CCSDS 231.0 (TC Synchronization and Channel Coding), the randomization is done before the encoding when BCH is enabled whereas if LDPC encoding is enabled, the randomization is done after the encoding. This has been changed in Yamcs version 5.5.4 - in versions 5.5.3 and earlier the randomization was always applied before the encoding (as per issue 3 of the CCSDS standard). If CUSTOM CLTU encoding is used, the custom encoder is responsible for the randomization - it can use this option or its own separate option for configuration.

skipRandomizationForVcs (list of integers) added in Yamcs 5.5.6
If randomizeCltu is true, this option can define a list of virtual channels for which randomization is not performed. This is not as per CCSDS standard which specifies that the randomization is enabled/disabled at the physical channel level.

cltuGeneratorClassName (string)
Required if cltuEncoding is CUSTOM. Specifies the name of the class which constructs the CLTU from the frame, if a custom format is required.

cltuGeneratorArgs
Optional if cltuEncoding is CUSTOM, ignored otherwise. Arguments to pass to the constructor for the CLTU generator classs.

virtualChannels (map)
Required. Used to specify the Virtual Channel specific configuration.

errorDetection (string)
One of `NONE` or `CRC16`. Specifies the error detection scheme used. If present, the last 2 bytes of the frame will contain an error control field. Default: `CRC16`

frameMaxRate (double)
maximum number of command frames to send per second. This option is specific to the UDP TC link.

For each Virtual Channel in the `virtualChannels` map, the following parameters can be used:

vcId (integer)
Required. The Virtual Channel identifier to be used in the frames. You can define multiple entries in the map with the same vcId, if the data is coming from different streams.

service (string)
Currently the only supported option is `PACKET` which is also the default.

commandPostprocessorClassName (string) and commandPostprocessorArgs (string)
Required if service=PACKET. Specfies the command post-processor and its configuration. See *Command Post-Processor* (page 60) for details.

stream (string)
Required. The stream on which the commands are received.

multiplePacketsPerFrame (boolean)
If set to true (default), Yamcs sends multiple command packets in one frame if possible (i.e. if the accumulated size fits within the maximum frame size and the commands are available when a frame has to be sent).

useCop1 (boolean)
If set to true, the COP-1 protocol is used for acknowledgment of TC frames.

clcwStream (string)
If COP-1 is enabled, this parameter configures the stream where the Command Link Control Words (CLCW) is read from.

initialClcwWait (integer)
If COP-1 is enabled, this specifies how many seconds to wait for the first CLCW.

cop1T1 (integer)
If COP-1 is enabled, this specifies the value in seconds for the timeout associated to command ac-

knowledgments. If the command frame is not acknowledged within that time, it will be retransmitted. The default value is 3 seconds.

cop1TxLimit (integer)
If COP-1 is enabled, this specifies the number of retransmissions for each un-acknolwedged frame before suspending operations.

bdAbsolutePriority (false)
If COP-1 is enabled, this specifies that the BD frames have absolute priority over normal AD frames. This means that if there are a number of AD frames ready to be uplinked and a TC with `cop1Bypass` flag is received (see below for an explanation of this flag), it will pass in front of the queue so ti will be the first frame uplinked (once the multiplexer decides to uplink frames from this Virtual Channel). This flag only applies when the COP-1 state is active, if the COP-1 synchronization has not taken place, the BD frames are uplinked anyway (because all AD frames are waiting).

tcQueueSize (integer)
This is used if COP-1 is not enabled, to determine the size of the command queue. Note that this is number of commands (not frames!). If the queue is full, the new commands will be rejected. Commands are taken from the queue by the multiplexer, according to the priority scheme defined below. Default: `10`.

errorDetection (string)
One of `NONE` or `CRC16`. Specifies the error detection scheme used for the virtual channel, overriding the setting at link level. This is not according to the CCSDS standard which specifies the frame error detection shall be configured at physical channel leve. If not specified (default), the setting at the link level will be used.

5.10.2.1 Priority Schemes

The multiplexing of command frames from the different Virtual Channels is done according to the defined priority scheme. The multiplexer is triggered by the availability of the uplink - when a command frame is to be uplinked it has to decide from which Virtual Channel it will release it.

`FIFO` means that the first frame received across all virtual channels will be the first one sent.

`ABSOLUTE` means that the frames will be sent according to the priority set on each Virtual Channel (set by the `priority` parameter). This means that as long as a high priority VC has commands to be sent, the lower priority VC will not release any command.

`POLLING_VECTOR` means that a polling vector will be built and each Virtual Channel will have the number of entries in the vector according to its priority. The multiplexing algorithm will cycle througugh the vector releasing the first command available. For example if there are two VCs VC1 with priority 2 and VC2 with priority 4, the polling vector will look like: [VC1, VC1, VC2, VC2, VC2, VC2]. This means that if both VCs have a high number of frames to be sent, the multiplexer will send 2 frames from VC1 followed by 4 from VC2 and then again. If however VC2 has only one frame to be sent, it will lose its other three slots for that cycle and the multiplexer will go back to sending two frames from VC1.

5.10.2.2 COP-1 Support

COP-1 is the protocol specified in CCSDS 232.1-B-2[74] for ensuring complete and correct transmission of TC frames. The protocol is using a sliding window principle based on the frame counter assigned by Yamcs to each uplinked frame.

The mechanism through which the on-board system reports the reception of commands is called Command Link Control Word (CLCW). This is a 4 byte word which is sent regularly by the on-board system to ground and contains the value of the latest received command counter and a few status bits. In Yamcs, we expect the CLCW to be made available on a stream (configured with the `clcwStream` parameter). The TM frame decoding can place the content of the OCF onto this stream. If the CLCW is sent as part of a regular TM packet, a StreamSQL statement like the following can be used:

[74] https://public.ccsds.org/Pubs/232x1b2e2c1.pdf

```
create stream clcw (clcw int)
insert into clcw select extract_int(packet, 12) as clcw from tm_realtime where extract_short(packet, 0) = 2080
```

The first statement creates the stream, and the second inserts 4 bytes extracted from offset 12 from all telemetry packets having the first 2 bytes equal with 2080.

If the `initialClcwWait` parameter is positive, at the link startup, Yamcs waits for that number of seconds for a CLCW to be received; once it is received, Yamcs will set the value of the ground counter (called `vS` in the spec) to the on-board counter value (called `nR` in the spec) received in the CLCW. That will ensure that the next command frame sent by Yamcs will contain the counter value expected by the on-board system.

If the `initialClcwWait` parameter is not positive (the value will be ignored) or if no CLCW has been received within the specified time, the synchronization has to be initiated manually via the user interface. This can be done either waiting again for a new CLCW, setting manually a value for `vS` (this requires the operator to know somehow what value the on-board system is expecting) or sending a command to the on-board system to force the on-board counter to the same value like the ground.

If the ground and on-board systems are not synchronized and a command is received, there are two possible outcomes:

- if the initialization process has been started (manually or at the link startup with the `initialClcwWait` parameter), the command will be put in a wait queue to be sent once the Synchronization took place.
- if the initialization process has not been started or has failed, the command will be rejected straight away with the NACK on the Sent acknowledgment.

AD, BD and BC frames

The CCSDS Standard distinguishes between three types of TC frames (the type is encoded in some bits in the frame primary header):

- AD frames contain normal telecommands and they are subjected to COP-1 transmission verification.
- BD frames contain normal telecommands but they are not subjected to COP-1 transmission verification.
- BC frames contain control commands generated by the ground COP-1 state machine and they are used to control the on-board state machine.

To send BD frames with Yamcs, you can use an attribute on the command called `cop1Bypass`. If the link finds this attribute set to true, it will send the command in a BD frame, bypassing the COP-1 verification. The BC frames are sent only by the COP-1 state machine and it is not possible to send them from the user.

The user interface allows also to deactivate the COP-1 and the user can opt for sending all the commands as AD frames or BD frames regardless of the cop1Bypass attribute.

6. Processors

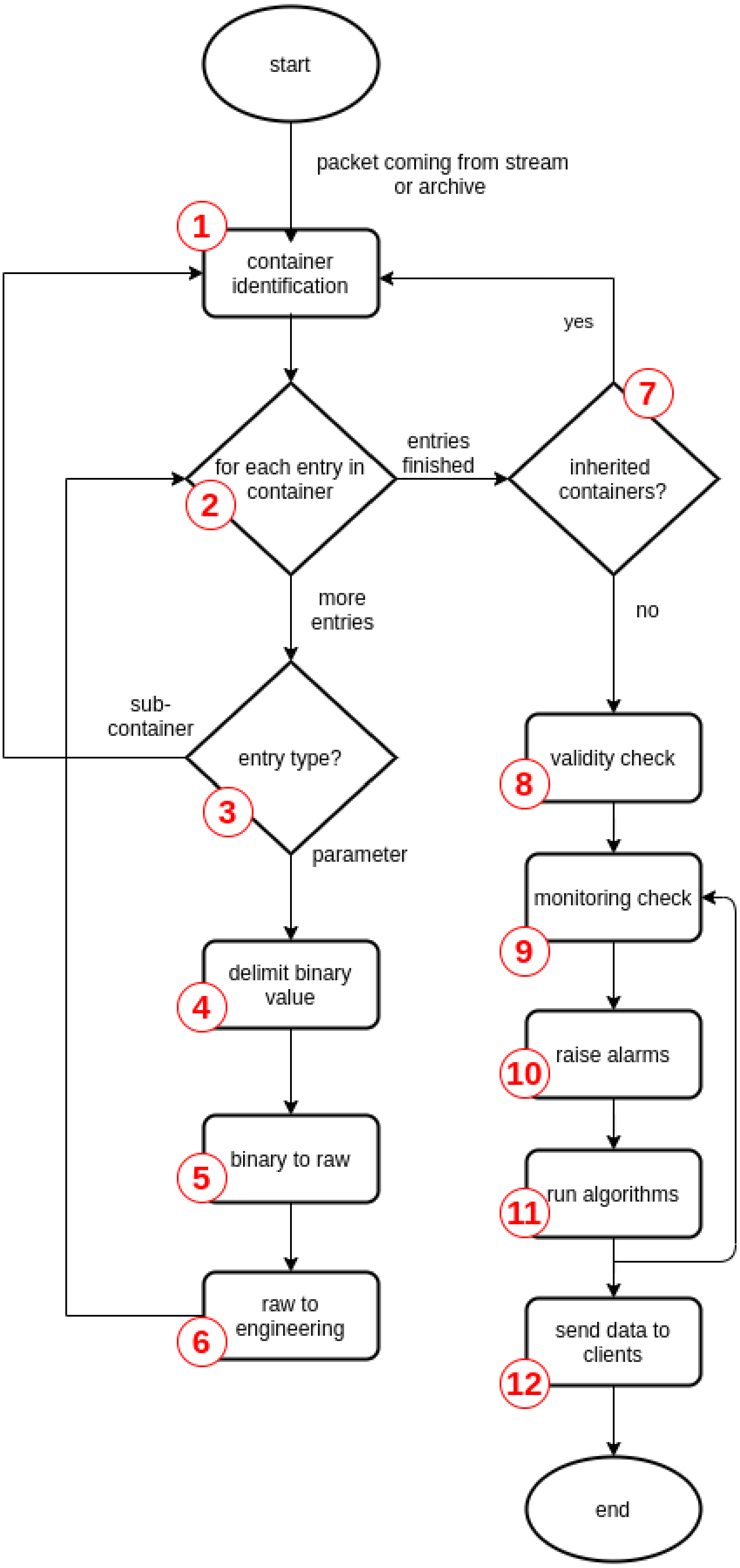
Yamcs processes TM/TC according to Mission Database definitions. Yamcs supports concurrent processing of parallel streams; one processing context is called *Processor*. Processors have clients that receive TM and send TC. Typically one Yamcs instance contains one realtime processor processing data coming in realtime and on-request replay processors, processing data from the archive. Internally, Yamcs creates a replay processors for tasks like filling up the Parameter Archive.

Each processor is composed of a set of services with varying functionality.

6.1 TM Packet Processing

This section describes how Yamcs processes the TM packets according to the MDB definitions. Note that when Yamcs receives a packet, it sends it first to a *Packet Pre-processor* (page 56) which assigns it a generation time. The generation time is saved in the archive and also used as generation time for all parameters extracted from that packet.

The figure below provides an overview of the steps involved, followed by a more detailed description of each step.



1. Container identification

When Yamcs has to process a TM packet it has first to know which MDB container it corresponds to. For realtime packets this is done based on the stream on which the packet is coming. Each stream has an attribute `rootContainer` (defined in `streamConfig -> tm` in `yamcs.<instance>.yaml`) which configures the container used for all packets coming on that stream. For historical reasons that attribute is optional; if not configured Yamcs will take the first container (when traversing the MDB tree) having no parent. For archive packets, that information is stored in the archive the first time the packet is received.

The processing returns to this step when processing sub-containers or inherited containers (see below).

2. Entry selection

Once a packet is matched to a container, Yamcs can proceed to extract the container entries. There are cases when, in order to improve the performance, Yamcs performs a partial retrieval - only a subset of the entries are processed. This for example is done as part of the XtceTmRecorder service when Yamcs does not want to extract parameters but only to identify the packet to its lowest sub-container; it is also done when performing a reply for the purpose of extracting a parameter or a set of parameters. The property that configures this behaviour is `config -> subscribeAll` in the processor configuration in `processor.yaml`. If the property is set to false, then only the parameter subscribed (and the dependent parameters) will be extracted.

3. Entry processing

A container has different types of entries:

- parameters - for these the processing continues with the steps 3-6 below.
- sub-containers - a new container processing context (bit offset starting from 0!) is created and the processing continues from step 1.
- array parameters (not shown in the figure) - these have their size either preset in the MDB or given by another parameter which has already been extracted from the packet. Yamcs will loop and extract the necessary number of array elements according to their data types in the steps 3-6 below. Note that there cannot be gaps between the elements of the array.
- aggregate parameters (not shown in the figure) - are parameters containing multiple members (like a struct in C). For each member of the aggregate Yamcs will extract the aggregate member according to its data type. As for the array elements, there can be no gap between aggregate members.
- indirect parameters (not shown in the figure) - are place-holders for other parameters - the exact parameter that will be extracted is determined by the value of another parameter proceeding it in the packet. Typically there is an id followed by some data, the data represents one parameter value and the id tells which parameter exactly the data represents.

For each entry, the MDB defines the position (in bits) in the packet of the start of the entry. This can be specified either absolute from the beginning of the packet or relative to the previous entry. Note that XTCE allows positions relative to the end of the packet but this is not supported by Yamcs.

4. Binary value delimitation

The start of the parameter in the packet is given by the offset in the container entry definition as explained above. The size in bits is given by the *Data Encoding* (or element/member Data Encoding for the arrays/aggregates). Some data encodings can be fixed in size (e.g. a 32 bit floating point number or a 4 bits integer), some of them can be variable size - typical examples are strings or binary. Finally there can be an user defined algorithm which can determine the size and also extract the raw value as explained below. The data decoding algorithms have to be implemented in Java (Javascript and Python are not supported)

5. Binary to raw value

Part of the data type processing is also extracting the raw value from the packet. The raw value is one of the usual types: boolean, signed/unsigned integer (max 64 bits), float (32 or 64 bits), string, binary, array or aggregate. Note that the binary type is an array of bytes but it is not an array parameter type. Using a binary parameter types instead of an array of 8 bit integers is more efficient and thus preferred in most cases.

6. Raw to engineering value

Next step in the processing is the conversion of the raw value to the engineering value. This is done using the calibration rule (if any) part of the **Parameter Type** MDB definition. A special case is an enumerated parameter type - the engineering value of such parameter is a special type called EnumeratedValue which has a dual integer/string representation. Other special type is an absolute time - the engineering value is a timestamp (resolution is millisecond). After the engineering value has been computed, Yamcs defers further processing until all entries have been extracted.

7. Container inheritance

After all entries from one container have been extracted, Yamcs proceeds to check if there is any inherited container which matches the condition. If there is, the container is processed starting with step 1. Note that the bit offset is not re-initialized to 0 as for sub-containers. It is considered that the inherited container contains the entries of the parent (already extracted) and thus any absolute position is counted from the beginning of the original packet.

8. Validity check

After all entries from the root container and all inherited containers to the deepest level have been extracted, Yamcs proceeds to perform checks on the extracted parameters. The first is checking against the validity range (if any) - if the check fails the parameter is declared as invalid and monitoring limits in the next step are not checked (no alarm raised either). If a parameter is invalid, it usually means that something went wrong with the transmission of the data.

9. Monitoring check

If a parameter has passed the validity checks, the monitoring checks are performed. This means checking a numeric parameter as being inside certain limits or checking an enumerated parameter having certain values. There is no monitoring check for boolean, string, binary, aggregate or array parameters. The monitoring checks can use contextual information - that means the limits checked depend on other parameters. The monitoring checks can be disabled by setting `config -> alarm -> parameterCheck` to false in processor.yaml

10. Alarm raising

If the alarm server is enabled (`config -> alarm -> parameterServer` in processor.yaml), alarm will be raised for all parameters which are determined by the previous step to be out of limits.

11. Algorithms

If there is any algorithm taking as input one of the parameters extracted, the value is provided to the algorithm. Depending on the algorithm definition, the algorithm is also run possibly producing more parameters. These parameters are also passed through the monitoring checks and alarms in step 9 and 10 (if they have defined limits).

12. Data distribution

Finally the list of all parameter values (those extracted from packet and those computed by algorithms) are distributed to all clients (displays, yamcs-web, etc).

6.2 Processor Configuration

The configuration of the different processor types can be found in `etc/processor.yaml`. The file defines a map whose keys are the processor types. The type is used to define a specific configuration used when creating the processor. In addition to its type, each processor has a unique name specified at the moment of creation.

The Yamcs processors are created in various ways:

- at startup by the *Processor Creator Service* (page 101). This is how typically the realtime processor is started. Note that here “realtime” is both the type and the name of the processor.
- by asking for archive data via the API with *dataSource = replay*. This will create a processor of type “ArchiveRetrieval”.

- the *Parameter Archive Service* (page 98) creates regularly processors of type “ParameterArchive” to build up the parameter archive.
- new processors of any type can be created via API. Yamcs Studio and Yamcs Web make use of this functionality to perform replays of data from the archive and they create processors of type “Archive”.

Note that the types “Archive”, “ParameterArchive” and “ArchiveRetrieval” are often hardcoded in the services that use those processor types so it is advisable not to change them in the processor.yaml. The user can define additional processor types for implementing custom functionality.

One current restriction is that all instances share the same processor types. It is not possible for example that the ParameterArchive processor type behaves differently in two different instances of the same Yamcs server.

Example of the realtime processor type configuration:

```
realtime:
    services:
      - class: org.yamcs.StreamTmPacketProvider
      ...
    config:
        subscribeAll: true
        recordInitialValues: true
        recordInitialValues: true
        maxTcSize: 4096

        alarm:
            parameterCheck: true
            parameterServer: enabled
            eventServer: enabled
            eventAlarmMinViolations: 1

        parameterCache:
            enabled: true
            cacheAll: true
            duration: 600
            maxNumEntries: 4096

        tmProcessor:
            ignoreOutOfContainerEntries: false
            expirationTolerance: 1.9
```

6.2.1 Options

services (list)

A list of services that are started together with the processor. The list is similar with the list of services used in the instance definitions. The reason is that originally (Yamcs v1) there were no instances but only processors and the data links were connected directly to them. The different available services are described in the subseqent chapters after this one.

The other options are under the *config* key and documented herebelow:

subscribeAll (boolean)

If true, all the services that provide parameters will provide all parameters starting at the processor creation. If set to false, the parameter are requested (subscribed) only when the external user asks for them (for example when opening a display, Yamcs Studio will subscribe to all parameters that are in the display). One service which can benefit of this is the XTCE TM processor: sometimes it is possible to extract only a selected list of parameters from packets and skip alltogether the packets for which no parameter is requested. The advantage is that there is less work to perform; the disatvantage is that no value is available when subscribing a parameter for the first time (e.g. when opening a display for the first time, there will be no value shown until a packet containing the parameters on the display will have arrived).

The providers are free to ignore this option and to provide more parameters than subscribed. This is for example the case for the XTCE TM processor when extracting parameters from a packet where the position of the entries is not absolute but relative to a previous entry. In this case the only way to extract a parameter in the middle or end of the packet is to extact all the parameters appearing in front.

recordInitialValues
The Mission database can contain initial (default) values for parameters. Enabling this option will cause an archive entry to be created at processor start with the values for all these parameters.

recordLocalValues
Local parameters are those known inside Yamcs and not provided by an external system. They are set by users via API calls. This option allows to record the values for these parameters each time they change.

maxTcSize (integer)
The maximum size of a telecommand packet. This value will set the maximum value regardless of the command definition in the Mission Database. There can be commands which have variable size arguments that do not specify a maximum size; this option will practically limit those cases to an overall maximum.

subscribeContainerArchivePartitions (boolean)
If set to true (default) the containers declared to be used as archive partition are subscribed by default in the processor. Otherwise the containers are only subscribed when a user subscribes to them or to a parameter contained in them. If alarms are enabled, the subcription to the parameters that can trigger alarms will also cause some container subscriptions. The only reason to switch this option off is for improving the performance when doing a archive retrieval that only extracts a few parameters. It is thus advisable to only configure it for the ArchiveRetrieval processor type. Note: the statistics shown on the yamcs-web instance home page contain the containers subscribed inside the currently selected processor. If no container is subscribed, only the root containers will be shown.

6.2.2 Alarm options

These options are defined under the config -> alarm.

parameterCheck (boolean)
If set to true, the parameters will be checked against the Mission Database defined limits. The users will receive the limit information as part of the parameter status. For example Yamcs Studio displays these parameters with a red or yellow border, depending on the severity of the limit. If set to false the limits will be ignored and all parameters will have the status unmonitored (equivalnet with having no limit defined in the Mission Database).

parameterServer (string)
Can be enabled or disabled. If enabled, an alarm server managing the alarm status of parameters will be started as part of the processor. This option requires the parameterCheck to be enabled. If disabled but the parameterCheck set to true, the parameters will still have their out of limit status associated but there will be no alarms generated.

eventServer (string)
Can be enabled or disabled. If enabled, an alarm server managing the alarm status of events will be started as part of the processor. This works similarly with the alarms for parameters - the severity of the event is used to derive the severity of the alarm. However because the events do not have a definition similar with the parameters in the Mission Database, the event source/type is used as a key for the alarm. That means that if a second event with the same source,type is being received as one that has already triggered an alarm, it is considered another occurence of the same alarm.

eventAlarmMinViolations (integer)
The number of occurences of a specific event (identified by its source and type) required to raise an alarm. By default it is 1. Note that the parameters do not have this setting because it is part of the Mission Database definition.

6.2.3 Parameter Cache options

These options are defined under the config -> parameterCache.

The processors can make use optionally of a parameter cache that stores the last values of parameters. The cache is used by Yamcs web to plot parameters which are not yet in the Parameter Archive.

Note that regardless of this cache there is always a last value cache which holds only the last known value for each parameter. The last value cache cannot be disabled.

The parameter cache can cause huge amounts of memory (RAM) to be consumed. The current implementation org.yamcs.parameter.ArrayParameterCache[75] tries to minimize the memory requirement by using arrays of primitive values instead of java objects but even then, the memory consumed can be significant. Updating the cache is also quite CPU intensive.

enabled (boolean)
If true, the parameter cache will be enabled.

cacheAll (boolean)
If true, the cache will store all parameter value regardless if there is any user requesting them or not. If false, the values are added to the cache only for the parameters requested by a user. Once a parameter is added to the cache, its values are always cached. This option can be used to reduce the amount of memory used by the cache with the inconvenience that first time retrieving the values of one parameter will not have them in the cache. Note that the option *subscribeAll* above is somehow similar - if that is set to false, then only some parameters will be available for cache even if this option is set to true.

duration: 600
How long in seconds the parameters should be kept in the cache. This value should be tuned according to the parameter archive consolidation interval.

maxNumEntries: 4096
How many values should be kept in the cache for one parameter.

6.2.4 TM (container) processing options

These options are defined under the config -> tmProcessor.

ignoreOutOfContainerEntries (boolean)
If set to false (default), when processing a TM packet, parameters whose position falls outside of the packet, will generate a warning. This option can be used to turn off that warning. Usually it is a sign of an ill-defined Mission Database and it is better to fix the Misison Database than setting this option.

expirationTolerance (double)
The Mission Database can define an expected rate in stream for packets (containers) - that means how often a packet is expected to be sent by the remote system. The rate in stream property will cause Yamcs to set an expiration time for the parameters extracted from that packet. The expiration of parameters is used to warn the operators that they are potentially looking at stale data in the displays. Yamcs will compute the expiration time as the rate in stream defined in the Mission Databae multiplied by this configuration option. The tolerenace is needed in order to avoid generating false expiration warnings.

maxArraySize (integer)
The maximum size of arrays extracted from TM packets. The arrays can be dynamically sized (meaning the size is given by a parameter in the packet) and this option configures the maximum size allowed. Default: `10000`.

[75] https://yamcs.org/javadoc/yamcs/org/yamcs/parameter/ArrayParameterCache.html

6.3 Alarm Reporter

Generates events for changes in the alarm state of any parameter on the specific processor. Note that this is independent from the actual alarm checking.

6.3.1 Class Name

org.yamcs.alarms.AlarmReporter[76]

6.3.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```
realtime:
  services:
    - class: org.yamcs.alarms.AlarmReporter
```

6.3.3 Configuration Options

source (string)
The source name of the generated events. Default: `AlarmChecker`

6.4 Algorithm Manager

Executes algorithms and provides output parameters.

6.4.1 Class Name

org.yamcs.algorithms.AlgorithmManager[77]

6.4.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```
realtime:
  services:
    - class: org.yamcs.algorithms.AlgorithmManager
      args:
        libraries:
          JavaScript:
            - "mdb/mylib.js"
```

6.4.3 Configuration Options

libraries (map)
Libraries to be included in algorithms. The map points from the scripting language to a list of file paths.

[76] https://yamcs.org/javadoc/yamcs/org/yamcs/alarms/AlarmReporter.html

[77] https://yamcs.org/javadoc/yamcs/org/yamcs/algorithms/AlgorithmManager.html

6.5 Local Parameter Manager

Manages and provides local parameters.

6.5.1 Class Name

org.yamcs.parameter.LocalParameterManager[78]

6.5.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```yaml
realtime:
  services:
    - class: org.yamcs.parameter.LocalParameterManager
```

6.6 Replay Service

Provides telemetry packets and processed parameters from the archive.

6.6.1 Class Name

org.yamcs.tctm.ReplayService[79]

6.6.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```yaml
Archive:
  services:
    - class: org.yamcs.tctm.ReplayService
```

6.6.3 Configuration Options

excludeParameterGroups (list of string)
Parameter groups to exclude from being replayed.

6.7 Stream Parameter Provider

Provides parameters received from the configured `param` stream.

6.7.1 Class Name

org.yamcs.tctm.StreamParameterProvider[80]

[78] https://yamcs.org/javadoc/yamcs/org/yamcs/parameter/LocalParameterManager.html
[79] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/ReplayService.html
[80] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/StreamParameterProvider.html

6.7.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```
realtime:
  services:
    - class: org.yamcs.tctm.StreamParameterProvider
      args:
        stream: "pp_realtime"
```

6.7.3 Configuration Options

streams (list of strings)
Required. The streams to read.

6.8 Stream TC Command Releaser

Sends commands to the configured `tc` streams.

The service supports sending commands to multiple streams depending on the command name. Each stream can be connected to a different data link, thus allowing Yamcs to control multiple targets concurrently.

The streams where the commands are sent to are defined as part of the *streamConfig section* (page 45) in the `yamcs.<instance>.yaml` instance configuration file.

6.8.1 Class Name

org.yamcs.StreamTcCommandReleaser[81]

6.8.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```
realtime:
  services:
    - class: org.yamcs.StreamTcCommandReleaser
```

6.8.3 Configuration Options

stream (string)
The stream to send commands to. This option is deprecated in favour of the stream configuration defined at instance level. Among others, that configuration is preferred because it allows having different streams for different instances, whereas the `processor.yaml` which defines this service is common for all instances.

6.9 Stream TM Packet Provider

Receives packets from `tm` streams and sends them to the processor for extraction of parameters.

This respects the root container defined as part of the `streamConfig` in *yamcs.yaml*.

[81] https://yamcs.org/javadoc/yamcs/org/yamcs/StreamTcCommandReleaser.html

6.9.1 Class Name

org.yamcs.StreamTmPacketProvider[82]

6.9.2 Configuration

This service is defined in `etc/processor.yaml`. Example:

```yaml
realtime:
  services:
    - class: org.yamcs.StreamTmPacketProvider
      args:
        streams: ["tm_realtime", "tm_dump"]
```

6.9.3 Configuration Options

streams (list of strings)

Required. The streams to read.

[82] https://yamcs.org/javadoc/yamcs/org/yamcs/StreamTmPacketProvider.html

7. Commanding

Yamcs supports XTCE concepts for commanding. Commands have constraints (preconditions) and verifiers (postconditions). The constraints are checked before issuing an command and the verifiers are run after the command has been issued to verify the different stages of execution.

In addition to the constraints/verifiers, Yamcs also implements the concept of command queue. This allows an operator to inspect commands of other users before being issued. It also allows to block completely commands from users during certain intervals (this effect can also be obtained with a constraint).

The commands and arguments are formatted to binary packets based on the XTCE definition.

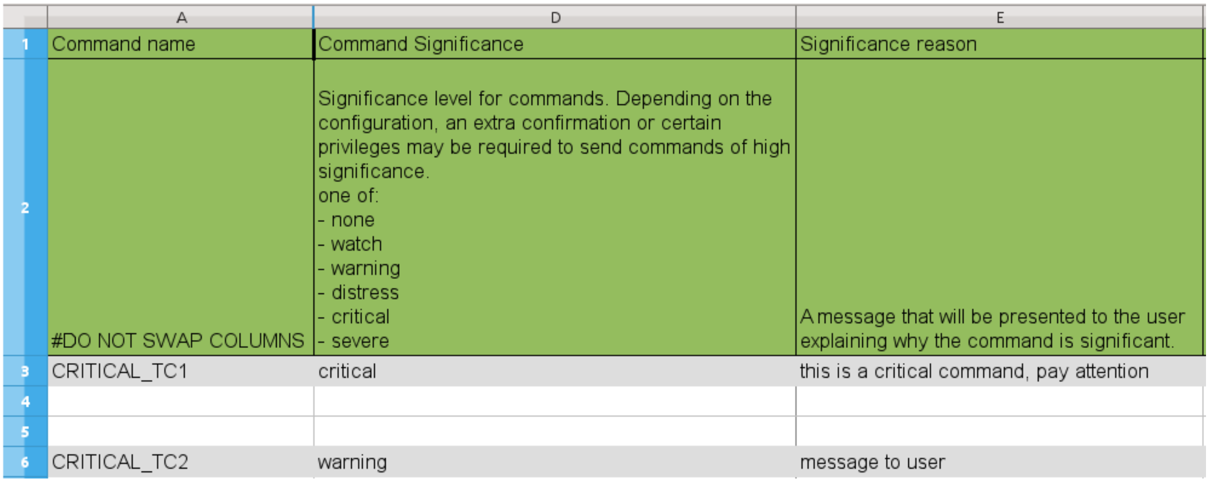
7.1 Command Significance

Yamcs uses the XTCE concept of command significance. Each command's significance can have one of this values none (default), watch, warning, distress, critical or severe.

In addition to the significance, the command has a message explaining why the command has the given significance.

Currently, Yamcs Server does not check or impose anything based on the significance of the command. In the future, the privileges may be used to restrict users that can send commands of high significance. However, currently the information (significance + reason) is only given to an external application (Yamcs Studio) to present it to the user in a suitable manner.

The command significance can be defined in the Excel Spreadsheet in the CommandOptions tab:



	A	D	E
1	Command name	Command Significance	Significance reason
2	#DO NOT SWAP COLUMNS	Significance level for commands. Depending on the configuration, an extra confirmation or certain privileges may be required to send commands of high significance. one of: - none - watch - warning - distress - critical - severe	A message that will be presented to the user explaining why the command is significant.
3	CRITICAL_TC1	critical	this is a critical command, pay attention
4			
5			
6	CRITICAL_TC2	warning	message to user

7.2 Command Queues

When a command is issued, it must first pass by a queue. Privileges are checked before the command is put into the queue, so if the user does not have the privilege for the given command, the command is rejected before even reaching the queue.

The available queues are defined in the file `etc/command-queue.yaml`.

```yaml
supervised:
  state: enabled
  minLevel: critical

default:
  state: enabled
```

If this file is absent, a default queue is created, equivalent to this configuration:

```yaml
default:
  state: enabled
```

Each queue has a name, a default state and optional conditions. Issued commands are offered to the first queue (in definition order) whose conditions matches the command.

The conditions are:

- **minLevel** (one of watch, warning, distress, critical or severe)
  Match only commands that are at least as significant as `minLevel`.
- **users** (list of usernames)
  Match only commands that are issued by one of the specified users.
- **groups** (list of group names)
  Match only commands that are issued by one of the specified groups.

The conditions `users` and `groups` are evaluated together: it suffices if the issuer matches with one of these two conditions. All other conditions must all apply, before a command can be matched to the queue.

At runtime, a queue can perform different actions on matched commands:

- **ACCEPT** (state: `enabled`)
  Matched commands are immediately released.
- **HOLD** (state: `blocked`)
  Matched commands are accepted into the queue but need to be manually released.
- **REJECT** (state: `disabled`)
  Matched commands are immediately rejected.

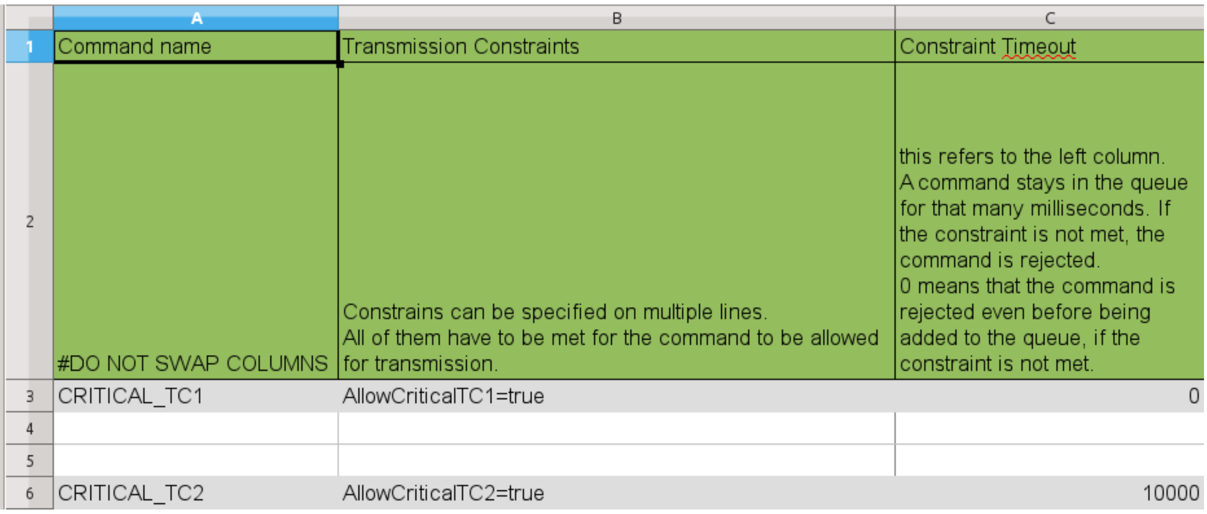
The queue action can be changed dynamically by users with the `ControlCommandQueue` privilege.

7.3 Transmission Constraints

When the is set to be released from the queue (either manually by an operator or automatically because the queue was in the Enabled state), the transmission constraints are verified.

The command constraints are conditions set on parameters that have to be met in order for the command to be releasd. There can be multiple constraints for each command and each constraint can specify a timeout which means that the command will fail if the constraint is not met within the timeout. If the timeout is 0, the condition will be checked immediately.

The transmission constraints can be defined in the Excel Spreadsheet in the CommandOptions tab.



	A	B	C
1	Command name	Transmission Constraints	Constraint Timeout
2	#DO NOT SWAP COLUMNS	Constrains can be specified on multiple lines. All of them have to be met for the command to be allowed for transmission.	this refers to the left column. A command stays in the queue for that many milliseconds. If the constraint is not met, the command is rejected. 0 means that the command is rejected even before being added to the queue, if the constraint is not met.
3	CRITICAL_TC1	AllowCriticalTC1=true	0
4			
5			
6	CRITICAL_TC2	AllowCriticalTC2=true	10000

Currently it is only possible to specify the transmission constraints based on parameter verification. This corresponds to Comparison and ComparisonList in XTCE. In the future it will be possible to specify transmission constraints based on algorithms. That will allow for example to check for specific values of arguments (i.e. allow a command to be sent if cmdArgX > 3).

8. Services

Yamcs functionality is modularised into different services, representing objects with operational state, with methods to start and stop. Yamcs acts as a container for services, each running in a different thread. Services carry out a specific function. Some services are vital to core functionality, others can be thought of as more optional and give Yamcs its pluggable nature.

Services appear at different conceptual levels:

- **Global services** provide functionality across all instances.
- **Instance services** provide functionality for one specific instance.

8.1 Global Services

8.1.1 HTTP Server

Embedded HTTP server that supports static file serving, authentication and API requests.

The HTTP Server is tightly integrated with the security system of Yamcs and serves as the default interface for external tooling wanting to integrate. This covers both server-to-server and server-to-user communication patterns.

The HTTP Server can be disabled when its functionality is not needed. Note that in this case also official external clients such as Yamcs Studio will not be able to connect to Yamcs.

8.1.1.1 Class Name

org.yamcs.http.HttpServer[83]

8.1.1.2 Configuration

This is a global service defined in `etc/yamcs.yaml`. Example:

```yaml
services:
  - class: org.yamcs.http.HttpServer
    args:
      port: 8090
      webSocket:
        writeBufferWaterMark:
          low: 32768
          high: 65536
      cors:
        allowOrigin: "*"
        allowCredentials: false
```

[83] https://yamcs.org/javadoc/yamcs/org/yamcs/http/HttpServer.html

8.1.1.3 Configuration Options

address (string)
The local address to which Yamcs will bind waiting for HTTP clients. If unset, Yamcs binds to a wildcard address.

port (integer)
The port to which Yamcs will bind waiting for HTTP clients. Default: `8090`

tlsCert (string or list of strings)
If specified, the server will be listening for TLS connections. TLS is used for encrypting the data.

In case the file is a bundle containing multiple certificates, the certificates must be ordered from leaf to root.

Multiple certificate files may also be provided as an array. Again, certificates must then be ordered from leaf to root, between the files and also between certificates within the files.

tlsKey (string)
Required if `tlsCert` is specified. The key to the certificate.

contextPath (string)
Path string prepended to all routes. For example, a contextPath of `/yamcs` will make the api available on `/yamcs/api` instead of the default `/api`. When using this property in combination with a reverse proxy, you should ensure that the proxy path matches with the context path because rewriting may lead to unexpected results.

Use this property only as a last resort. For multi-hosting situations, it is preferable to serve Yamcs from the root of its own subdomain.

zeroCopyEnabled (boolean)
Indicates whether zero-copy can be used to optimize non-SSL static file serving. Default: `true`

maxContentLength (integer)
Maximum allowed length of request bodies. This is applied to all non-streaming API requests. Default: `65536`

Some routes may specify a custom `maxBodySize` option, in which case the maximum of the two values gets applied.

nThreads (integer)
Configure the number of threads that handle HTTP requests. The value `0` resolves to two times the number of CPU cores. Default: `0`

reverseLookup (boolean)
If enabled, hostnames instead of IP addresses are used to identify clients. Use of this option may trigger name service reverse lookups. Default: `false`

webSocket (map)
Configure WebSocket properties. Detailed below. If unset, Yamcs uses sensible defaults.

cors (map)
Configure cross-origin resource sharing for the HTTP API. Detailed below. If unset, CORS is not supported.

8.1.1.3.1 WebSocket sub-configuration

maxFrameLength (integer)
Maximum frame length in bytes. Default: `65536`

writeBufferWaterMark (map)
Water marks for the write buffer of each WebSocket connection. When the buffer is full, messages are dropped. High values lead to increased memory use, but connections will be more resilient against unstable networks (i.e. high jitter). Increasing the values also help if a large number of messages are generated in bursts. The map requires keys `low` and `high` indicating the low/high water mark in bytes.

Default: `{ low:  32768, high:  65536}`

8.1.1.3.2 CORS sub-configuration CORS (cross-origin resource sharing) facilitates use of the API in client-side applications that run in the browser. CORS is a W3C specification enforced by all major browsers. Details are described at https://www.w3.org/TR/cors/. Yamcs simply adds configurable support for some of the CORS preflight response headers.

Note that the embedded web interface of Yamcs does not need CORS enabled, because it shares the same origin as the HTTP API.

allowOrigin (string)
Exact string that will be set in the `Access-Control-Allow-Origin` header of the preflight response.

allowCredentials (boolean)
Whether the `Access-Control-Allow-Credentials` header of the preflight response is set to true. Default: `false`

8.1.2 Process Runner

Runs an external process. If this process exits this Yamcs service stops too unless a `restart` option is configured and applicable.

8.1.2.1 Class Name

org.yamcs.ProcessRunner[84]

8.1.2.2 Configuration

This is a global service defined in `etc/yamcs.yaml`. Example:

```
services:
  - class: org.yamcs.ProcessRunner
    args:
      command: "bin/simulator.sh"
```

8.1.2.3 Configuration Options

command (string or string[])
Required. Command (and optional arguments) to run.

directory (string)
Set the working directory of the started subprocess. If unspecified, this defaults to the working directory of Yamcs.

logLevel (string)
Level at which to log stdout/stderr output. One of `INFO`, `DEBUG`, `TRACE`, `WARN`, `ERROR`. Default: `INFO`

logPrefix (string)
Prefix to prepend to all logged process output. If unspecified this defaults to '`[COMMAND]`'.

restart (string)
When to start a new process if the original process exits. One of `always`, `on-success`, `on-failure` or `never`. Default: `never`.

successExitCode (integer or integer[])
Exit codes of the subprocess that are considered successful. This is used to evaluate the appropriate `restart` behaviour. Default: `0`.

[84] https://yamcs.org/javadoc/yamcs/org/yamcs/ProcessRunner.html

8.1.3 TSE Commander

This service allows dispatching commands to Test Support Equipment (TSE). The instrument must have a remote control interface (Serial, TCP/IP) and should support a text-based command protocol such as SCPI.

8.1.3.1 Class Name

org.yamcs.tse.TseCommander[85]

8.1.3.2 Configuration

This is a global service defined in `etc/yamcs.yaml`. Example:

```
services:
  - class: org.yamcs.tse.TseCommander
```

8.1.3.3 Configuration Options

telnet (map)
: **Required.** Configure Telnet properties.

Example: `{ port: 8023 }`

tc (map)
: **Required.** Configure properties of the TC link.

Example: `{ port: 8135 }`

tm (map)
: **Required.** Configure properties of the TM link.

Example: `{ host: localhost, port: 31002 }`

This service reads further configuration options from a file `etc/tse.yaml`. This file defines all the instruments that can be commanded. Example:

```
instruments:
  - name: tenma
    class: org.yamcs.tse.SerialPortDriver
    args:
      path: /dev/tty.usbmodem14141
      # Note: this instrument does not terminate responses.
      # Use a very short timeout to compensate (still within spec)
      # responseTermination: "\n"
      responseTimeout: 100

  - name: simulator
    class: org.yamcs.tse.TcpIpDriver
    args:
      host: localhost
      port: 10023
      responseTermination: "\r\n"

  - name: rigol
    class: org.yamcs.tse.TcpIpDriver
    args:
      host: 192.168.88.185
      port: 5555
      responseTermination: "\n"
```

There are two types of drivers. Both drivers support these base arguments:

[85] https://yamcs.org/javadoc/yamcs/org/yamcs/tse/TseCommander.html

responseTermination (string)
The character(s) by which the instrument delimits distinct responses. Typically `\n` or `\r\n`. This may be left unspecified if the instrument does not delimit responses.

commandSeparation (string)
The character(s) that indicates when the command will generate multiple *distinct* responses (delimited by `responseTermination`). For most instruments this should be left unspecified.

responseTimeout (integer)
Timeout in milliseconds for a response to arrive. Default: `3000`

requestTermination (string)
Character(s) to append to generated string commands. This is typically used for adding newline characters with make the instrument detect a complete request.

Set this to null if you do not want to disable request termination.

The default value is driver-specific. For the TCP/IP driver it defaults to `\n` whereas for the Serial Port driver, it is unset.

interceptors (list of maps)
Adds an interceptor chain where each interceptor must be an implementation of org.yamcs.tse.Interceptor[86]. Interceptors are executed top-down on these events:

1. A new command is about to be issued. The interceptor can inspect it, or make final changes. The input is in the form of a raw byte array and includes any request termination characters (if applicable).

2. A non-null response was received. The interceptor can inspect it, or make adjustments before handing it over to the next interceptor. Only at the end of the chain, the response bytes are interpreted by the TSE Commander. Note that the response bytes do **not** include the response termination characters (if any), because the driver already strips them off while delimiting messages from the incoming stream.

Yamcs ships with one standard interceptor which you can add to an instrument's configuration if you want to enable logging of its command and response messages:

```yaml
- name: myinstrument
  class: org.yamcs.tse.TcpIpDriver
  args:
    ...
    interceptors:
      - class: org.yamcs.tse.LoggingInterceptor
```

In addition each driver supports driver-specific arguments:

8.1.3.3.1 TCP/IP

host (string)
Required. The host of the instrument.

port (integer)
Required. The TCP port to connect to.

8.1.3.3.2 Serial Port

path (string)
Required. Path to the device.

baudrate (number)
The baud rate for this serial port. Default: 9600

dataBits (number)
The number of data bits per word. Default: 8

[86] https://yamcs.org/javadoc/yamcs/org/yamcs/tse/Interceptor.html

parity (string)
The parity error-detection scheme. One of odd or even. By default parity is not set.

8.1.3.4 Mission Database

The definition of TSE string commands is done in space systems resorting under /TSE. The /TSE node is added by defining org.yamcs.xtce.TseLoader[87] in the MDB loader tree. Example:

```
mdb:
  - type: org.yamcs.xtce.TseLoader
    subLoaders:
      - type: sheet
        spec: mdb/tse/simulator.xls
```

The instrument name in etc/tse.yaml should match with the name of the a sub space system of /TSE.

The definition of commands and their arguments follows the same approach as non-TSE commands but with some particularities:

- Each command should have either QUERY or COMMAND as its parent. These abstract commands are defined by the org.yamcs.xtce.TseLoader[88].
  - QUERY commands send a text command to the remote instrument and expect a text response. The argument assignments command and response must both be set to a string template that matches what the instrument expects and returns.
  - COMMAND commands send a text command to the remote instrument, but no response is expected (or it is simply ignored). Only the argument assignment command must be set to a string template matching what the instrument expects.
- Each TSE command may define additional arguments needed for the specific command. In the definition of the command and response string templates you can refer to the value of these arguments by enclosing the argument name in angle brackets. Example: an argument n can be dynamically substituted in the string command by referring to it as <n>.
- Additionally you can instruct Yamcs to extract one or more parameter values out of instrument response for a particular command by referring to the parameter name enclosed with backticks. This parameter should be defined in the same space system as the command and use the non-qualified name. The raw type of these parameters should be string.

To illustrate these concepts with an example, consider this query command defined in the space system /TSE/simulator:

Command name	Assignments	Arguments

get_identification
(parent: QUERY)
	command=*IDN?
	response=`identification`	

When issued, this command will send the string *IDN? to the instrument named simulator. A string response is expected and is read in its entirity as a value of the parameter /TSE/simulator/identification.

The next example shows the definition of a TSE command that uses a dynamic argument in both the command and the response string templates:

[87] https://yamcs.org/javadoc/yamcs/org/yamcs/xtce/TseLoader.html
[88] https://yamcs.org/javadoc/yamcs/org/yamcs/xtce/TseLoader.html

Command name	Assignments	Arguments

get_battery_voltage
(parent: QUERY)
	command=:BATTERY<n>:VOLTAGE?
	response=`battery_voltage<n>`
		n (range 1-3)

When issued with the argument 2, Yamcs will send the string :BATTERY2:VOLTAGE? to the remote instrument and read back the response into the parameter /TSE/simulator/battery_voltage2. In this simple case you could alternatively have defined three distinct commands without arguments (one for each battery).

Note: When using the option commandSeparation, the response argument of the command template should use the same separator between the expected responses. For example a query of :DATE?;:TIME? with command separator ; may be matched in the MDB using the pattern: `date_param`;`time_param` (regardless of the termination character).

8.1.3.5 Telnet Interface

For debugging purposes, this service starts a telnet server that allows to directly relay text-based commands to the configured instruments. This bypasses the TM/TC processing chain. Access this interface with an interactive TCP client such as telnet or netcat.

The server adds additional SCPI-like commands which allow to switch to any of the configured instruments in a single session. This is best explained via an example:

```
$ nc localhost 8023
:tse:instrument rigol
*IDN?
RIGOL TECHNOLOGIES,DS2302A,DS2D155201382,00.03.00
:cal:date?;time?
2018,09,14;21,33,41
:tse:instrument tenma
*IDN?
TENMA72-2540V2.0
VOUT1?
00.00
:tse:output:mode hex
VOUT1?
30302E3030
```

In this session we interacted with two different instruments (named rigol and tenma). The commands starting with :tse were directly interpreted by the TSE Commander, everything else was sent to the selected instrument.

8.1.4 Replication Server

The replication server facilitates the communication between *Replication Master* (page 106) and *Replication Slave* (page 107). The master and slaves defined with the tcpRole *server* will register to this component to be called when an external tcp client connects. Multiple master and slaves from different Yamcs instances in the same Yamcs server will register to the same replication server.

A remote slave when connecting will send a request message indicating the instance and the transaction it wants to start the replay with. The replication server will forward the request to the registered local master which will immediately start the replay.

A remote master when connecting to the replication server will send a wakeup message indicating the instance of the slave. The replication server will redirect the message to the registered local slave which in turn will send a request to the master indicating the transaction start.

8.1.4.1 Class Name

org.yamcs.replication.ReplicationServer[89]

8.1.4.2 Configuration

This service is defined in `etc/yamcs.yaml`. Example:

```
services:
    - class: org.yamcs.replication.ReplicationServer
      args:
         port: 8099
         tlsCert: /path/to/server.crt
         tlsKey: /path/to/server.key
         maxTupleSize: 131072
```

8.1.4.3 Configuration Options

port (integer)
Required The port to listen for TCP connections.

tlsCert (string or list of strings)
If specified, the server will be listening for TLS connections. TLS is used for encrypting the data, client certificates are not supported. If TLS is enabled, all connections have to be encrypted, the server does not support TLS and non-TLS connections simultaneously.

In case the file is a bundle containing multiple certificates, the certificates must be ordered from leaf to root.

Multiple certificate files may also be provided as an array. Again, certificates must then be ordered from leaf to root, between the files and also between certificates within the files.

tlsKey (string)
Required if `tlsCert` is specified. The key to the certificate.

maxTupleSize (integer)
Used for the slaves with tcpRole = server - configures the maximum size of the serialized tuples received from the master. If the serialized tuples are larger than this size, this limit has to be increased otherwise the tuples cannot be transferred. Default 131072 (128KB).

8.2 Instance Services

8.2.1 Alarm Recorder

Records alarms. This service stores the data coming from one or more streams into a table `alarms`.

8.2.1.1 Class Name

org.yamcs.archive.AlarmRecorder[90]

[89] https://yamcs.org/javadoc/yamcs/org/yamcs/replication/ReplicationServer.html
[90] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/AlarmRecorder.html

8.2.1.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```yaml
services:
  - class: org.yamcs.archive.AlarmRecorder

streamConfig:
  alarm:
    - alarms_realtime
```

With this configuration alarms emitted to the `alarms_realtime` stream are stored into the table `alarms`.

8.2.2 Command History Recorder

Records command history entries. This service stores the data coming from one or more streams into a table `cmdhist`.

8.2.2.1 Class Name

org.yamcs.archive.CommandHistoryRecorder[91]

8.2.2.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```yaml
services:
  - class: org.yamcs.archive.CommandHistoryRecorder

streamConfig:
  event:
    - cmdhist_realtime
    - cmdhist_dump
```

With this configuration events emitted to the `cmdhist_realtime` or `cmdhist_dump` stream are stored into the table `cmdhist`.

8.2.3 Event Recorder

Records events. This service stores the data coming from one or more streams into a table `events`.

8.2.3.1 Class Name

org.yamcs.archive.EventRecorder[92]

8.2.3.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

[91] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/CommandHistoryRecorder.html

[92] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/EventRecorder.html

```
services:
  - class: org.yamcs.archive.EventRecorder

streamConfig:
  event:
    - events_realtime
    - events_dump
```

With this configuration events emitted to the `events_realtime` or `events_dump` stream are stored into the table `events`.

8.2.4 CCSDS TM Index

Creates an index for the CCSDS Space Packets (*CCSDS 133.0-B-1 <https://public.ccsds.org/Pubs/133x0b1c2.pdf>*) based on the sequence count in the primary header. The index allows to see per APID the available packets in the archive. The main use of such index is to detect when packets are missing. It can be combined with user defined scripts that request missing data from remote systems (if such systems exist that record data in the user specific setup).

The configuration allows to define a list of tm streams where the packets are read from. The packets on those streams have to be CCSDS space packets. This service does not use the Mission Database for interpreting the packets, it just reads the primary header from the binary data. If the packet legnth is less than 7 bytes, it is discarded.

The index can be visualised in the Yamcs web interface in the Archive -> Overview. It is denoted as “Completeness” and contains one timeline bar for each APID.

8.2.4.1 Class Name

org.yamcs.archive.CcsdsTmIndex[93]

8.2.4.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
  - class: org.yamcs.archive.IndexServer
    streams: ["tm-realtime", "tm_dump"]
```

8.2.4.3 Configuration Options

streams (list of strings)

The streams to index. When unspecified, all `tm` streams defined in `streamConfig` are indexed.

8.2.5 Parameter Archive Service

The Parameter Archive stores time ordered parameter values. The parameter archive is column-oriented and is optimized for accessing a (relatively small) number of parameters over longer periods of time. Data is stored in fixed duration time intervals, each interval covering a length of 2^23 milliseconds (~139 minutes).

An interval has always to be processed or reprocessed in full - this means if one data point is added in the interval, the full 139 minutes of data have to be reprocessed.

Intervals are further split into segments such that each segment cannot contain more than a configurable maximum number of samples. This is done in order to limit the number of samples stored in memory when

[93] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/CcsdsTmIndex.html

rebuilding an interval. A parameter that comes at high frequency will be split into multple segments whereas for one that comes at low frequency there will be only one segment in each interval.

The parameters are grouped such that the samples of all parameters from one group have the same timestamp. For example all parameters extracted from one TM packet have usually the same timestamp and are part of the same group. A special case is the aggregate parametes: these are decomposed into the individual members if scakar types but all values are belonging to the same group and thus the aggregate can be rebuilt even though the members are stored separately.

8.2.5.1 Class Name

org.yamcs.parameterarchive.ParameterArchive[94]

8.2.5.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```yaml
services:
  - class: org.yamcs.parameterarchive.ParameterArchive
    args:
      realtimeFiller:
        enabled: true
        flushFrequency: 300  #seconds
      backFiller:
        #warmupTime: 60 seconds default warmupTime
        enabled: true
        schedule: [{startInterval: 10, numIntervals: 3}]
```

This configuration enables the realtime filler, and in addition the backFiller fills the archive 10 intervals in the past, 3 intervals at a time.

```yaml
services:
  - class: org.yamcs.parameterarchive.ParameterArchive
    args:
      realtimeFiller:
        enabled: false
      backFiller:
        enabled: true
        warmupTime: 120
        schedule:
          - {startInterval: 10, numIntervals: 3}
          - {startInterval: 2, numIntervals: 2, frequency: 600}
```

This configuration does not use the realtime filler, but instead performs regular (each 600 seconds) backfillings of the last two intervals. It is the configuration used in the ISS ground segment where due to regular (each 20-30min) LOS (loss of signal), the archive is very fragmented and the only way to obtain continuous data is to perform replays.

8.2.5.3 General Options

maxSegmentSize (integer)

The ParameterArchive stores data in segments, each segment storing multiple samples of the same parameter. This option configures the maximum segment size. The parameter archive accumulates data in memory to fill the segments, in parallel for all parameters. This option affects thus the memory consumed when the parameter archive is being filled. The segment size is limited by the duration of an interval, a segment cannot be larger than 2^23 milliseconds (approximatevely 139 minutes). Default: `5000`

[94] https://yamcs.org/javadoc/yamcs/org/yamcs/parameterarchive/ParameterArchive.html

8.2.5.4 Backfiller Options

These options appear under the `backFiller` key.

enabled (boolean)
If true the backfiller is enabled and executes according to the schedule. Default: `true`

warmupTime (integer)
When the backfiller performs a replay to fill a data interval, the replay will start this number of seconds before the interval start. This is sometimes required for algorithms that aggregate data, to be able to have all the input data necessary to produce the output. Default: `60`

monitorStreams (string[])
The list of tm streams that will be monitored to check for new data. If the list is empty, no stream will be monitored and the archive will be rebuilt according to the schedule defined below. Default: all the tm streams defined in the yamcs.instance.yaml streamConfig section. The backfiller will check the generation time of the packet received on the monitoring streams and will mark that interval as `dirty`. As soon as the `streamUpdateFillFrequency` timer (see below) expires, a new filling task is started for that interval.

streamUpdateFillFrequency (integer)
Valid if the `monitorStreams` is not empty, configures how often in seconds the fillup based on the stream monitoring is started. The fillup only starts if new data has been received on the streams. Default `600`.

schedule (list of maps)
This option contains a list of schedules configuring when the parameter archive runs. This is used when the back filler does not monitor any input stream and instead rebuilds the archive according to a schedule (even if there was maybe no new data received). Each map in the list has the following keys:

startInterval (integer) ** Required.** when a backfiller starts, it starts processing with this number of intervals in the past.

numIntervals (integer) ** Required.** how many intervals to process at one time

frequency (integer)

compactFrequency (integer)
After how many backfilling tasks to compact the underlying RocksDB database. Because the backfiller removes the previous data, RocksDB will have lots of thombstones to skip over when reading. Compacting will get rid of the thombstones. Compacting improves the reading at the expense of writing speed. -1 means that no compaction will be performed (RocksDB merges by itself files, and that also gets rid of the thombstones). Default value: 5

8.2.5.5 Realtime filler Options

enabled (boolean)
If true the backfiller is enabled and executes according to the schedule. Default: `true`

processorName (String)
The name of the processor used to receive realtime data. Default: `realtime`

sortingThreshold (integer) milliseconds
When receiving realtime data, the realtime filler builds up data in memory. In order to know that data can be written to the archive (whole segments at once) the filler needs to know that no data can be received into the old segments. This option configures in milliseconds the amount of acceptable unsorting - that is each new data timestamp which is older than the previous received data timestamp, will be accepted as long as the difference is not bigger than this. This option is interpreted at the level of parameter group; For example having multiple streams of TM packets (a stream understood as an ordered sequence of packets not necessarily a Yamcs stream) with different timestamps is not a problem as long as each stream has its monotonic increasing time. Note also the option `pastJumpThreshold` below. Default: `1000`

pastJumpThreshold (integer) seconds
When processing data and the time jumps in the past with more than this number of seconds, the realtime filler will flush all the segments to disk and start from scratch. Default `86400`.

numThreads (integer)
The realtime filler will compress and flush the segments to disk in background. This option configures how many threads should be used for that operation. The default is the total number of CPUs of the system minus 1.

8.2.6 Parameter Recorder

Records parameters. This service stores the data coming from one or more streams into a table `pp`. The term *pp* stands for processed parameter. These are parameters that typically are processed by an external system before being recorded in Yamcs. It is also used to store system parameters that are generated by Yamcs itself.

Note: Parameters extracted from packets are usually not stored in `pp`. Instead Yamcs provides a different service called the *Parameter Archive* (page 98) which is specially optimized for data retrieval.

8.2.6.1 Class Name

org.yamcs.archive.ParameterRecorder[95]

8.2.6.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
  - class: org.yamcs.archive.ParameterRecorder

streamConfig:
  param:
    - pp_realtime
    - sys_param
```

With this configuration both system parameters and processed parameters coming from the `pp_realtime` stream are stored into the table `pp`.

8.2.6.3 Configuration Options

streams (list of strings)
The streams to record. When unspecified, all `param` streams defined in `streamConfig` are recorded.

8.2.7 Processor Creator Service

Creates persistent processors owned by the system user.

8.2.7.1 Class Name

org.yamcs.ProcessorCreatorService[96]

[95] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/ParameterRecorder.html
[96] https://yamcs.org/javadoc/yamcs/org/yamcs/ProcessorCreatorService.html

8.2.7.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
  - class: org.yamcs.ProcessorCreatorService
    args:
      name: realtime
      type: realtime
```

8.2.7.3 Configuration Options

name (string)
  Required. The name of the processor

type (string)
  Required. The type of the processor

config (string)
  Configuration string to pass to the processor

8.2.8 Replay Server

This service handles replay requests of archived data. Each replay runs with a separate processor that runs in parallel to the realtime processing.

8.2.8.1 Class Name

org.yamcs.archive.ReplayServer[97]

8.2.8.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
  - class: org.yamcs.archive.ReplayServer
```

8.2.9 System Parameters Service

Collects system parameters from any Yamcs component at a frequency of 1 Hz. Parameter values are emitted to the `sys_var` stream.

8.2.9.1 Class Name

org.yamcs.parameter.SystemParametersService[98]

[97] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/ReplayServer.html

[98] https://yamcs.org/javadoc/yamcs/org/yamcs/parameter/SystemParametersService.html

8.2.9.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
  - class: org.yamcs.parameter.SystemParametersService
    args:
      provideJvmVariables: true
```

8.2.9.3 Configuration Options

provideJvmVariables (boolean)
When set to `true` this service will create a few system parameters that allows monitoring basic JVM properties such as memory usage and thread count. Default: `false`

8.2.10 XTCE TM Recorder

Records XTCE TM sequence containers. This service stores the data coming from one or more streams into a table `tm`. The tm table has a column called `pname` which stands for packet name. The main task of this service is to assign the value for that column; all the other colums will come directly from the tm stream as provided by the data links.

The pname is a fully qualified name of a matching XTCE containter. In the XTCE hierarchy some containers have a flag `useAsArchivingPartition` (this flag is an Yamcs extension to XTCE). That flag is used to determine the container that will give its name to the packet when saved into the tm table - the name of the lowest level matching container with this flag set is chosen as the pname. If no container matches, then the name of the root container will be used.

8.2.10.1 Class Name

org.yamcs.archive.XtceTmRecorder[99]

8.2.10.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
  - class: org.yamcs.archive.XtceTmRecorder

streamConfig:
  tm:
    - tm_realtime
    - tm_dump
```

With this configuration containers coming from both the tm_realtime and tm_dump streams are stored into the table `tm`.

8.2.10.3 Configuration Options

streams (list of strings)
The streams to record. When unspecified, all `tm` streams defined in `streamConfig` are recorded.

[99] https://yamcs.org/javadoc/yamcs/org/yamcs/archive/XtceTmRecorder.html

8.2.11 Time Correlation Service

Correlates (synchronizes) time between a free running on-board clock and ground.

It receives samples `(obt, ert)` where:

- `obt` - onboard time considered to be a counter running based on an on-board computer clock.
- `ert` - Earth Reception Time - the time when the signal has been received on the ground - it is typically provided by a ground station.

It takes into account the parameters:

- `onboardDelay` - It covers any delay happening on-board (sampling time, radiation time)
- `tof` - time of flight - the time it takes for the signal to reach the ground. This can be fixed or computed by dynamically interpolating from data provided by a flight dynamics system.

Assuming that:

ob_time = ert - (tof + onboardDelay)

the service will compute `m` = gradient and `c` = offset such that:

ob_time = m * obt + c

Using the computed gradient and offset, the free running obt can be correlated to the ground time. The process has to be repeated each time the on-board computer resets to 0 (this typically happens when the computer reboots). This method can compensate for a linear drift of the on-board clock.

The determination of the gradient and offset is done using the least squares method.

The number of samples used for computing the coefficients is configurable and has to be minimum 2.

The ground time `ert` being provided by a ground station (and not by Yamcs), is considered to be accurate enough for the required purpose.

Note about accuracy: the main usage of this service is to timestamp the telemetry received from the on-board system. Yamcs keeps such timestamps at milliseconds resolution. However the service keeps internally the time at picosecond resolution so theoretically it can be used to achieve better than millisecond accuracy. In practice this is not so easy: it requires an accurate on-board clock, an accurate ground-station clock, a good time of flight estimation taking into account various effects (ionospheric, tropospheric delays, etc). All the dynamic delays have to be incorporated into the time of flight estimation.

8.2.11.1 Accuracy and validity

Once the coefficients have been calculated, for each new sample received a deviation is calculated as the delta between the OBT computed using the coefficients and the OBT which is part of the sample (after adjusting for delays). The deviation is compared with the accuracy and validity parameters:

- If the deviation is greater than `accuracy` but smaller than `validity`, then a recalculation of the coefficients is performed based on the last received samples.
- If the deviation is greater than `validity` then the coefficients are declared as invalid and all the samples from the buffer except the last one are dropped. The time returned by `getTime()` will be invalid until the required number of new samples is received and the next recalculation is performed.

8.2.11.2 Verify Only Mode

If the on-board clock is synchronized via a different method, this service can still be used to verify the synchronization.

The method `verify(TmPacket pkt)` will check the difference between the packet generation time and the expected generation time (using `ert - delays`) and in case the difference is greater than the validity, the packet will be changed with the local computed time and the flag {@link TmPacket#setLocalGenTime()} will also be set.

8.2.11.3 Usage

To use this service the preprocessor (or other mission specific service) will adds samples using the addSample(long, Instant) each time it receives a correlation sample from on-board. How the on-board system will send such samples is mission specific (for example the PUS protocol defines some specific time packets for this purpose).

The preprocessor can then use the method getTime(long obt) to get the time corresponding to the obt or call timestamp(long obt, TmPacket pkt) to timestamp the packet. The second method will timestamp the packet with a time derived from the ert if the service is not synchronized. A corresponding flag will be set on the packet so it can be distingushed in the archive.

8.2.11.4 Time of flight estimation

As explained above, the correlation process requires the estimation of the time of flight between the spacecraft and the ground station. This can be configured to a static value or dynamically computed based on the user supplied polynomials on time intervals. The HTTP API[100] can be used to add the intervals and corresponding polynomials.

8.2.11.5 Class Name

org.yamcs.time.TimeCorrelationService[101]

8.2.11.6 Configuration

This service is defined in etc/yamcs.(instance).yaml. Example:

```
services:
    - class: org.yamcs.time.TimeCorrelationService
      name: tco0
      args:
          onboardDelay: 0.0
          useTofEstimator: false
          defaultTof: 0.0
          accuracy: 0.1
          validity: 0.2
          numSamples: 3
```

8.2.11.7 Configuration Options

onboardDelay (double)
the on-board delay in seconds used to compute the on-board transmission time from the earth reception time. The default value is 0 seconds.

useTofEstimator (boolean)
Flag to enable or disable time of flight estimator service. The default value is false. Enable time of flight estimator service when it is required to dynamically compute the time of flight.

defaultTof (double)
The default time of flight in seconds. This value is used if the tof estimator does not return a value because no interval has been configured.

accuracy (double)
The accuracy in seconds. See above for an explanation on how this value is used. The default value is 0.1 (100 milliseconds).

[100] https://docs.yamcs.org/yamcs-http-api/time-correlation/add-time-of-flight-intervals/

[101] https://yamcs.org/javadoc/yamcs/org/yamcs/time/TimeCorrelationService.html

validity (double)

The validity in seconds. See above for an explanation on how this value is used. The default value is 0.2 (200 milliseconds).

numSamples (integer)

How many samples to collect before computing the correlation coefficients. It has to be minimum 2; the default value is 3.

8.2.12 Replication Master

Replicates data streams to remote servers. Works both in TCP server and TCP client mode. In TCP server mode, it relies on the *Replication Server* (page 95) to provide the TCP connectivity. In TCP client mode, it connects to a list of slaves specified in the configuration.

The master works by storing stream of tuples serialized in memory mapped files org.yamcs.tctm.replication.ReplicationFile[102]. Each tuple receives an 64 bit incremental transaction id. In addition to the tuple data, there are some metadata transactions storing information about the streams and allowing the data to be compressed. For example a parameter tuple has the potentially very long qualified parameter names as column names, these are only stored in the metadata and replaced in the data by 32 bit integers. The serialization mechanism is the same used for serializing tuples in the stream archive but there is no distinction between the key and the value.

The replication files are append only (except for a header which contains the number of tuples stored) and contain a configurable number of tuples. The maximum size of the file is also configurable so a new file is created either when the maximum number of transactions has been reached or when the maximum size of the file has been reached.

The replication slaves are responsible for keeping track of their last received transaction id. In both TCP client and server mode, the slaves are sending to the master the first transaction id and the master starts replaying from there. In case the slave has not connected for a long time, the first transaction may be in one of the deleted files. The master will start sending from the first transaction available.

New in version 5.6.1: the master will regularly send time messages in order to keep the connection alive if there is no data. The slave can optionally use the time message to update the local mission time, synchronizing it to the master.

8.2.12.1 Class Name

org.yamcs.replication.ReplicationMaster[103]

8.2.12.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```yaml
services:
    - class: org.yamcs.replication.ReplicationMaster
      args:
          tcpRole: client
          pageSize: 500
          maxPages: 500
          streams: ["tm_realtime", "tm2_realtime"]
          maxFileSizeKB: 102400
          expirationDays: 7
          fileCloseTimeSec: 300
          slaves:
              - host: "localhost"
                port: 8099
```

(continues on next page)

[102] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/replication/ReplicationFile.html

[103] https://yamcs.org/javadoc/yamcs/org/yamcs/replication/ReplicationMaster.html

(continued from previous page)

```
                instance: "node2"
                enableTls: false
          reconnectionInterval: 5000
```

8.2.12.3 Configuration Options

tcpRole (string)
Required One of client or server.

maxPages (integer)
The number of pages of the replication file. The replication file header contains an index allowing to access the start of each page. Thus more pages, the faster is to jump to a given transaction but the larger the header. Since seeking a transaction is only performed when a slave connects, it is not critical that the search is very fast. The total number of transactions in one file is maxPages * pageSize. Default: 500

pageSize (integer)
The number of transactions on one page. Default: 500

streams (list of strings)
The list of streams that will be replicated. The replication file will contain multiplexed data from these streams in order in which the data is generated. The connected slaves will receive data from all streams but they may filter it out locally.

maxFileSizeKB (integer)
Maximum size in KB of the replication file. Default 102400 (e.g. the maximum file size will be 100MB).

fileCloseTimeSec (integer)
How many seconds to keep a file open after being accessed by a slave. Default: 300.

expirationDays (double)
How many days to keep the replication files before removing them. Default: 7.

slaves (list of maps)
Required if the tcpRole is *client*. The list of slaves to connect to. Each slave is specified as a host/port and the slave instance name. In addition, TLS (encrypted connections) can be specified for each slave individually using the enableTls option. The replication master will connect to the replication server on the remote host/port and will send a Wakeup message containing the salve instance name; the replication server will then redirect the connection to the corresponding replication slave if one has registered for the given instance.

reconnectionIntervalSec (integer)
If the tcpRole is *client* this configures how often in seconds the replication master will try to connect to the salve if the connection is broken. A negative value means that no reconnection will take place.

timeMsgFreqSec (integer)
Added in version 5.6.1. How often (in seconds) should send the time messages. Default: 10

8.2.13 Replication Slave

The slave counterpart to the *Replication Master* (page 106). It receives serialized tuple data from the master and injects it in the local stream. Works both in TCP server and TCP client mode. In TCP server mode, it relies on the *Replication Server* (page 95) to provide the TCP connectivity. In TCP client mode, it connects to the master defined in the configuration.

The slave keeps track of the id of the last transaction received from the master in a local text file `yamcs-data/<instance>/replication/slave-lastid.txt`. Each time the connection to the master is established, it sends a request containing the last transaction id +1. The master will start replaying data from that transaction. If the replication slave does not find the file at startup, it will receive all the data that the master has.

There can be two or replication slaves running for the same instance, connected to two different masters.

To avoid an infinite message flood caused by a miss-configuration whereby a slave receives and inserts into a stream the data which was extracted from the same stream, each incoming messages contains a 32 bit `instance id`. This is the id of the instance where the message has originated from. If a slave receives a message with its own instance id it will discard it and not insert it into the stream. The instance id is calculated as a hash code from the `<serverId>.<instanceName>`. The serverId is by default the hostname but can be changed in yamcs.yaml.

New in version 5.6.1: the master will regularly send time messages in order to keep the connection alive if there is no data. The slave can optionally use the time message to update the local mission time, synchronizing it to the master.

8.2.13.1 Class Name

org.yamcs.replication.ReplicationSlave[104]

8.2.13.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```
services:
    - class: org.yamcs.replication.ReplicationSlave
      args:
          tcpRole: client
          masterHost: localhost
          masterPort: 8099
          masterInstance: node1
          enableTls: false
          reconnectionIntervalSec: 30
          streams: ["tm_realtime", "sys_param"]
          lastTxFile: "slave-lastid.txt"
```

8.2.13.3 Configuration Options

tcpRole (string)
: **Required** One of client or server.

masterHost (string)
: **Required** if the tcpRole is *client*. The hostname of the master. Not relevant if the tcpRole is *server*.

masterPort (integer)
: **Required** if the tcpRole is *client*. The port of the master. Not relevant if the tcpRole is *server*.

masterInstance (string)
: **Required** if the tcpRole is *client*. The instance of the master. When working in *server* tcp mode, the instance on which the master is configured determines the data which will be passed to the slave. If two masters try to connect to the same slave, only the first connection will be accepted.

enableTls (boolean)
: **Required** Used when tcpRole is *client*. If true, a TLS connection will be attempted. The server provided certificate will be checked against the trustStore in yamcs/etc/ directory. If the tcpRole is *server* the usage or not of TLS is determined by the configuration of the *Replication Server* (page 95).

reconnectionIntervalSec (integer)
: If the tcpRole is *client* this configures how often in seconds the slave will try to connect to the master if the connection is broken. A negative value means that no reconnection will take place. Default: 30

[104] https://yamcs.org/javadoc/yamcs/org/yamcs/replication/ReplicationSlave.html

streams (list of strings)
The list of streams that will be processed. The master may send data from other streams but they will be filtered out.

lastTxFile (String)
The name of file where the slave will keep track of the last transaction id received from the server. It defaults to the `<service-name>-lastid.txt`

maxTupleSize (integer)
Used in case the tcpRole = client - configures the maximum size of one message received from the master. If the serialized tuples are larger than this size, this limit has to be increased otherwise the tuples cannot be transferred. Default 131072 (128KB).

timeoutSec (float)
Added in version 5.6.1. Timeout (in seconds) for detecting broken connections. If no message is received in this time from the master, the connection will be closed. Even if there is no data, the master sends a time message at configurable intervals. Default: 30.

updateSimTime (boolean)
Added in version 5.6.1. If true, update the simulation time with the time received from the master in the time messages, allowing to synchronize the mission time between the master an the slave. This only works if the SimulationTimeService is configured on the same instance with this service. The time0 will be set to 0 at the service startup. The messages received regularly from the master contain the triplet (localTime, missionTime, speed) and will be used to call the methods setSimElapsedTime(long javaTime, long simElapsedTime) and the setSpeed(double speed) in the SimulationTimeService. The synchronization relies on the fact that the local (UNIX) times are synchronized between master and slave. This has to be ensured at the system level (e.g. using NTP). Default: false

8.2.14 CCSDS File Delivery Protocol (CFDP)

This service implements the CCSDS File Delivery Protocol class 1 (unreliable transfer also called unacknowledged) and class 2 (reliable transfer also called acknowledged).

Class 3 and 4 (transfers via one or more waypoints) are not supported.

The service uploads and downloads files between a spacecraft (or a remote device) and a Yamcs bucket. In the description below, the entity that sends the file is called the Sender and the entity that receives the file is called the Receiver.

The protocol specification can be found in CCSDS 727.0-B-5[105] The following description summarizes the specs and provide details on the parts implented/not implemented by this service.

The upload/download works by splitting the file into segments and uploading/downloading each segment individually (usually embedded as part of a TC/TM packet). The transmission is preceded by an metadata PDU (Protocol Data Unit) and finished with an EOF PDU. The Receiver will send the Finished PDU to let the Sender know that all PDUs have been received.

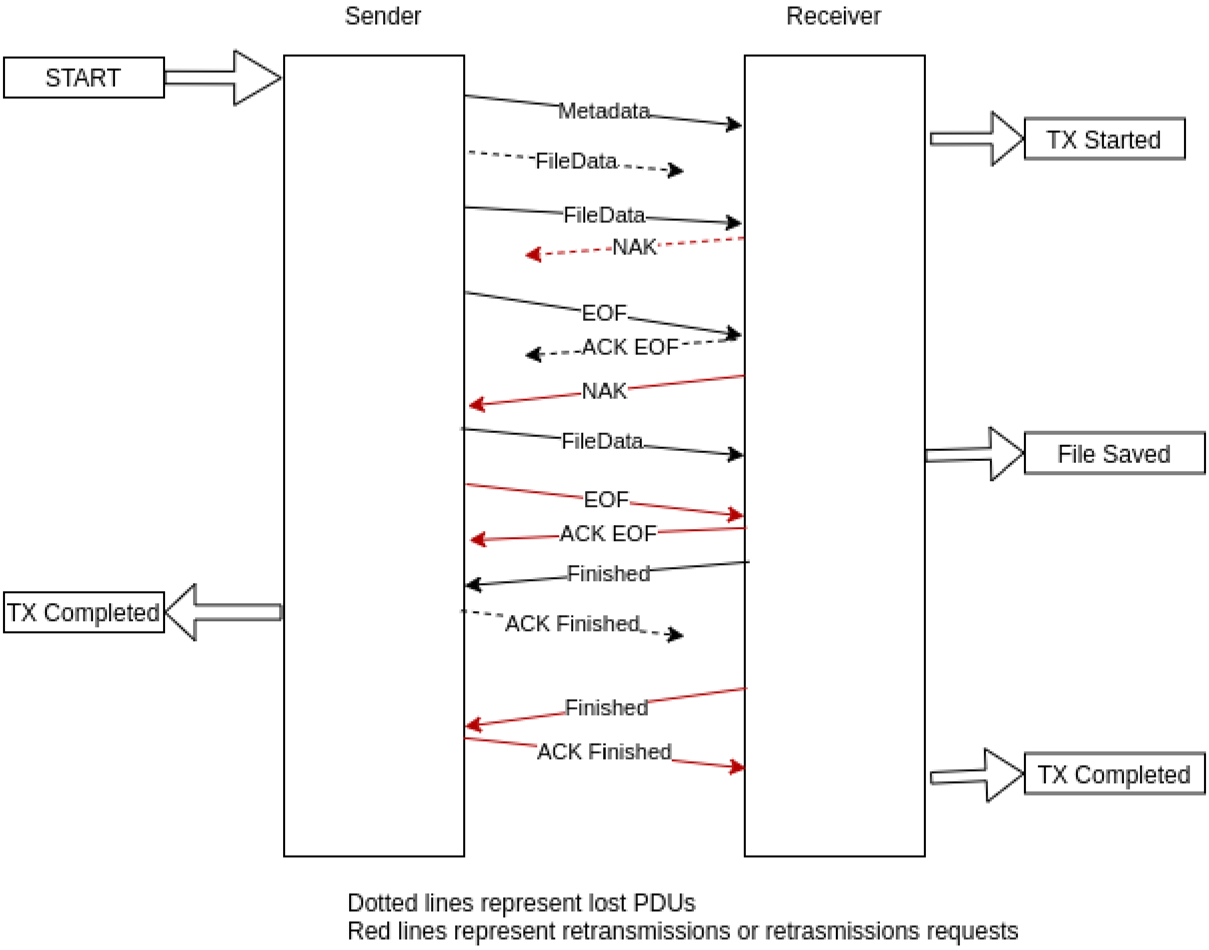
The class 1 (unreliable transfer) will upload/download all the segments without the possibility of retransmission. The EOF is not acknowledged by the Receiver. The Issue 5 of the CFDP standard introduces an option “Closure Requested” which requests the class 1 Receiver to send a Finished PDU upon receiving all the data (or when the canceling the transfer). The Finished PDU is not acknowledged by the Sender. This option is useful when the underlying communication protocol is reliable.

For class 2 (reliable transfer) transfers, the Receiver can indicate missing metadata or data by sending NAK PDUs. In this mode, the Receiver has to acknowledge the EOF PDU and the Sender has to acknowledge the Finished PDU. Sending a PDU that requires acknowledgment will start a timer. When the timer expires, if the acknowledgment has not been received, the PDU is resent and this is done until a count reaches a maximum defined value. Finally if the count has reached its maximum value and the acknowledgment has still not been received, a fault condition is triggered which may cause the transfer to be abandoned, canceled or suspended.

[105] https://public.ccsds.org/Pubs/727x0b5.pdf

A diagram of the operations for class 2 is presented in the figure below. Note that the Receiver operates in immediate NAK mode; it sends a NAK as soon as it receives the FileData PDU (containing a file segment) and detects a missing segment.

Note also that the file is available on the Receiver before the transfer is completed by the reception of the Finished ACK PDU.



The CFDP transfers can be suspended and resumed. Suspending means that no PDU is sent out but incoming PDUs are still processed. The timers are deactivated. Upon resuming, the timers are restarted and their counts reset to 0. For example if at the time of the suspension, an EOF has been sent 2 times out of 5, after the transfer is resumed, the EOF sending is again starting with 0 out of 5. This allows suspending the transfer when the limit has been reached and resume the transfer at a later moment without changing the state.

Several peculiarities and limitations of the implementation can be noted:

- The NAK PDUs issued by Sender always contain the beginning of the file up to filling up the PDU with data. Unless the file is very large and with lots of small gaps, a NAK PDU will contain all the missing data at the given point.
- The Receiver will overwrite the list of segments to resend with the list received in the latest NAK.
- Keep Alive PDU and Prompt PDU are not used.
- Filestore requests are not supported.
- User operations (proxy, directory, remote status) as per chapter 6 of the CCSDS 727.0-B-5 are not supported.
- Remote suspend/resume operations are not supported. Note that local suspend/resume operations are supported; this means that suspending a transfer has to be done concurrently on this

service and remotely with a different mechanism (e.g. sending a telecommand).

8.2.14.1 Usage

The service produces PDUs as per CCSDS specification. The PDUs are written/read to/from Yamcs streams. How the PDUs are sent to/from the spacecraft is mission specific.

An example on how to use the streams to embed the CFDP PDUs into CCSDS packets can be seen in the cfdp example (the most interesting bit is in `src/main/yamcs/etc/extra_streams.sql`).

8.2.14.2 Class Name

org.yamcs.cfdp.CfdpService[106]

8.2.14.3 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

```yaml
services:
   - class: org.yamcs.cfdp.CfdpService
     name: cfdp0
     args:
         sequenceNrLength: 4
         maxPduSize: 512
         incomingBucket: "cfdpDown"
         eofAckTimeout: 3000
         eofAckLimit: 3
         sleepBetweenPdus: 1000
         localEntities:
           - name: id1
             id: 11
             bucket: bucket1
           - name: id2
             id: 12
         remoteEntities:
           - name: target1
             id: 5
           - name: target2
             id: 7
             bucket: bucket3
         senderFaultHandlers:
           AckLimitReached: suspend
         receiverFaultHandlers:
           AckLimitReached: suspend
```

8.2.14.4 Configuration Options

name
The name of the service - used in the API calls. If multiple CfdpServices are used, this has to contain a different value for each service. By default it is “CfdpService”.

inStream (string)
The name of the stream where the CFDP PDUs are read from. Default: `cfdp_in`

outStream (string)
The name of the stream where the CFDP PDUs are written. Default: `cfdp_out`

incomingBucket (string)
The name of the bucket where the CFDP incoming files are saved if no specific ones are defined per local or remote entity. Default: `cfdpDown`

[106] https://yamcs.org/javadoc/yamcs/org/yamcs/cfdp/CfdpService.html

localEntities (map)

A list of entity definitions used to give names to the local (Yamcs) entity identifiers as well as to configure which bucket is used for storing the files received for that entity. The names can be used in the REST calls when initiating transfers. The list has to contain all identifiers which will be used by the remote system to send files. If a PDU is received to an identifier not in this map, the PDU will be dropped and no transaction will be started. The `bucket` is optional; if missing, the file will be saved into the bucket specified for the remote entity and if that is missing too in the general bucket configured with the `incomingBucket`.

remoteEntities (map)

A list of entity definitions used to give names to the remote (spacecraft) entity identifiers. The names can be used in the REST call when initiating transfers. The list has to contain all identifiers which will be used by the remote system to send files. If a PDU is received from an identifier not in this map, the PDU will be dropped and no transaction will be started. The list can contain also a bucket name used if the matching local entity does not define a bucket. In the example above if a file is downlinked having source (spacecraft) id = 7 and destination (Yamcs) id = 12, it will end up in bucket3.

entityIdLength (integer)

The length in bytes of the entity id for the outgoing CFDP transfers. The entity id and the sequence number represent the CFDP transaction identifier - it is encoded in the header of all the CFDP PDUs. These lengths determine the size of the PDU header. For the incoming transfers the remote peer specifies the lengths. Default: `2`

sequenceNrLength (integer)

The length in bytes of the sequence number for the outgoing CFDP transfers. Default: `4`

maxPduSize (integer)

The maximum length of the PDU is used by the sender to determine how to split the file into segments (segment size = PDU size - header size). For the incoming transfers the peer specifies the PDU size. Default `512`

sleepBetweenPdus (integer)

The time in milliseconds used by the sender to wait in between sending two successive PDUs. This together with the PDU determine the uplink data rate. The data rate has to match the maximum uplink speed as well as the receiver expected data rate. No mechanism is implemented for auto-tuning the uplink rate.

inactivityTimeout (integer)

The time in milliseconds used by both the sender and receiver to check for inactivity. The timer is active on the receiver until EOF has been received and on class 2 sender after EOF has been sent (while waiting for the Finished PDU). If the timer expires, the InactivityDetected event will be trigggred ad the transaction may be cancelled or suspended (depending on the configuration of the fault handler for InactivityDetected event). Default: `10000` (10 seconds).

eofAckTimeout (integer)

Valid for class 2 transfers; the time in milliseconds used by the sender to wait for the EOF PDU acknowledgment. The sender sends the EOF PDU to indicate that it has completed transmitting the file. It expects to receive an acknowledged indicating the reception of the EOF PDU (not of the file!, the Finished PDU is used for that). The EOF PDU is retransmitted if no acknowledgment has been received in this time. Default: `3000` (3 seconds).

eofAckLimit (integer)

Valid for class 2 transfers; the number of times to retry sending the EOF PDU before declaring a fault. Zero means that only one PDU will be sent (no retry). Negative value means no limit. Default: `5`.

finAckTimeout (integer)

The time in milliseconds used by the receiver to wait for the FIN PDU acknowledgment. The receiver sends the Finished PDU to indicate that the file has been received or that a fault has been encountered. The receiver expects the sender to acknowledge reception of this PDU and will retransmit the PDU if no acknowledgment has been received in this time. Default: `10000` (10 seconds)

finAckLimit (integer)

The number of times to retry sending the Finished PDU before declaring a fault. Zero means that only one PDU will be sent (no retry). Negative value means no limit. Default: `5`.

immediateNak (boolean)
Valid for class 2 transfers; if true, the receiver will send NAK PDUs as soon as a missing segment is detected. The NAK PDU contains the list of segments that are missing at the receiver side. If the EOF PDU has not been received, the NAK PDU covers only the segments missing up to the last one recieved. If this parameter is false, the receiver will only send NAK PDUs after the EOF PDU has been received. In this case the NAK PDU will contain all the missing segments. Default: `true`

nakTimeout (integer)
Valid for class 2 transfers; used by the receiver as the time interval between two successive NAK PDUs, assuming the data has not been recovered. Default: `5000`

nakLimit (integer)
Valid for class 2 transfers; the number of times to send a NAK PDU with no data recovered before declaring a fault. A value of 1 means that one NAK is sent and if no data is recovered within the nakTimeout milliseconds, a fault will be declared. Zero or negative value means no limit. Default: `-1`

senderFaultHandlers (map)
A definitions of the actions to be taken when the sender encounters different faults. The definitions are in the form of `conditionCode -> action` map. The possible condition codes are: AckLimitReached, KeepAliveLimitReached, InvalidTransmissionMode, FilestoreRejection, FileChecksumFailure, FileSizeError, NakLimitReached, InactivityDetected, InvalidFileStructure, CheckLimitReached and UnsupportedChecksum. The possible actions are: suspend, cancel or abandon. Suspend means the transfer will be suspended and can be resumed later (for example an ack limit reached may be caused by the lost of communication with the spacecraft and the transfer can be resumed when the communication is established again). Cancel means that the remote peer is notified that the transaction is canceled. Abandon means to abort the transaction without notifying the peer. Note that the error can be generated locally or recieved from the peer in a FIN PDU.

receiverFaultHandlers (map)
Similar with `senderFaultHandlers` but applies when the service works as Receiver (i.e. for downlinks).

maxNumPendingDownloads (integer)
The maximum number of allowed concurrent downloads. If this limit is reached, any PDU that would start a new dowload is dropped and an event message generated. Default: `100`

maxNumPendingUploads (integer)
The maximum number of allowed concurrent uploads. If this limit is reached, the new uploads are queued. Default: `10`

directoryTerminators (list)
When starting an upload to a directory (folder), the CFDP service will append the object name to the directory name. To know if the destination is a folder (and not a file), the end character is compared with the terminators in this list. Default: `["/", ":", "\\"]`

allowConcurrentFileOverwrites (boolean)
If this option is true, when starting an upload, the CFDP service verifies if an upload witht the same destination filename is ongoing or queued and will raise an error. This is done in order to avoid overwriting the same destination file in case of multiple files are uploaded from the yamcs-web. Default: `true`

pendingAfterCompletion (integer)
Number of milliseconds to keep the transaction in memory after completion. During this time, the PDUs received belonging to this transaction are still answered. Default: 600000 (10 minutes)

8.2.15 CFS Event Decoder

Decodes cFS[107] events. This service translates binary cFS telemetry packets into Yamcs events.

[107] https://cfs.gsfc.nasa.gov/

The packets are filtered by message id (first 2 bytes of the header).

The structure of the event packets is as defined in the CFE_EVS_LongEventTlm_Payload struct[108]. The structure had different names in older versions of cFS.

The field `EventType` is used to derive the event severity:

- value 3 is considered severity `ERROR`
- value 4 is considered severity `CRITICAL`
- all the other values are considered severity `INFO`

8.2.15.1 Class Name

org.yamcs.tctm.cfs.CfsEventDecoder[109]

8.2.15.2 Configuration

This service is defined in `etc/yamcs.(instance).yaml`. Example:

8.2.15.3 Configuration Options

msgIds ([integer])
The message ids that will be considered as events. This argument is required.

byteOrder (string):
The byte order of the event telemetry packet. Default:`BIG_ENDIAN`

charset (string):
The charset used to decode the text string. Default: `US-ASCII`

appNameMax (integer):
The size of the app name in bytes. Default: `20`

eventMsgMax (integer):
The size of the event message string in bytes. Default: `122`

streams ([string]):
The streams to process for events. Not required. If no stream is provided, all telemetry streams of type `tm` are used (these are configured in the instance configuration file uder the `streamConfig` section).

[108] https://github.com/nasa/cFE/blob/main/fsw/cfe-core/src/inc/cfe_evs_msg.h#L1235

[109] https://yamcs.org/javadoc/yamcs/org/yamcs/tctm/cfs/CfsEventDecoder.html

9. Security

Yamcs includes a security subsystem which allows authenticating and authorizing users. Authentication is the act of identifying the user, whereas authorization involves determining what privileges this user has.

Once authorized, the user may be assigned one or more privileges that determine what actions the user can perform. Yamcs distinguishes between system privileges and object privileges.

9.1 System Privileges

A system privilege is the right to perform a particular action or to perform an action on any object of a particular type.

ControlProcessor
Allows to control any processor.

CreateInstances
Allows to create instances.

ModifyCommandHistory
Allows to modify command history.

ControlCommandClearances
Allows to clear users for commanding.

ControlCommandQueue
Allows to manage command queues.

GetMissionDatabase
Allows to read Mission Database definitions.

ChangeMissionDatabase
Allows online changes to Mission Database definitions.

ReadAlarms
Allows to read alarms.

ControlAlarms
Allows to manage alarms.

ControlArchiving
Allows to manage archiving properties of Yamcs.

ReadLinks
Allows to read link state.

ControlLinks
Allows to control the lifecycle of any link.

ControlServices
Allows to manage the lifecycle of services.

ManageAnyBucket
Provides full control over any bucket (including user buckets).

ReadEvents
Allows to read any event.

ReadThreads
Allows to read thread information.

WriteEvents
Allows to manually create events.

WriteTables
Allows to manually add records to tables.

ReadTables
Allows to read tables.

Note: Yamcs plugins may support additional system privileges.

9.2 Object Privileges

An object privilege is the right to perform a particular action on an object. The object is assumed to be identifiable by a single string. The object may also be expressed as a regular expression, in which case Yamcs will perform pattern matching when doing authorization checks.

Command
Allows to issue a specific command.

CommandHistory
Allows access to the command history of a specific command.

ManageBucket
Allow control over a specific bucket.

ReadAlgorithm
Allows to read a specific algorithm.

ReadBucket
Allow readonly access to a specific bucket.

ReadPacket
Allows to read a particular packet.

ReadParameter
Allows to read a particular parameter.

Stream
Allow to read and emit to a specific stream.

WriteParameter
Allows to set the value of a specific parameter.

Note: Yamcs plugins may support additional object privileges.

9.3 Superuser

A user may have the attribute `superuser`. Such a user is not subject to privilege checking. Any check of any kind will automatically pass. An example of such a user is the `System` user which is used internally by Yamcs on some actions that cannot be tied to a specific user. The `superuser` attribute may also be assigned to end users if the AuthModule supports it.

9.4 AuthModules

9.4.1 LDAP AuthModule

The LDAP AuthModule supports authentication of users via the LDAP protocol.

It first searches for the distinguished name that matches a submitted username, and then attempts a bind using the submitted password.

This module can also be chained to the *Kerberos AuthModule* (page 119) or *SPNEGO AuthModule* (page 120) modules in order to add user attributes to a user that logged in via Kerberos or Kerberos SPNEGO.

9.4.1.1 Class Name

org.yamcs.security.LdapAuthModule[110]

9.4.1.2 Configuration Options

host (string)
Required. The LDAP host

userBase (string)
Required. The search base for users.

Example: `ou=people,dc=example,dc=com`

port (integer)
The LDAP port. Default: 389 for unencrypted connections, otherwise 636.

tls (boolean)
If `true` the LDAP connection will be encrypted. Default: `false`

user (string)
The bind DN that Yamcs should use to search user properties. If unspecified Yamcs will attempt to do an anonymous bind. On many LDAP installations an anonymous bind does not give sufficient access to user information.

password (string)
The password matching the bind DN.

attributes (map)
Configure which LDAP attributes are to be considered. If unset, Yamcs uses defaults that work out of the box with many LDAP installations.

9.4.1.2.1 Attributes sub-configuration

name (string)
The name of the account name attribute. This is used to search a DN within the `userBase` as well as to map to the Yamcs account name. Default: `uid`.

email (string or string[])
The name of the email attribute. If multiples are defined, they are tried in order. Default: `[mail, email, userPrincipalName]`.

displayName (string or string[])
The name of the display name attribute. If multiples are defined, they are tried in order. Default: `cn`.

[110] https://yamcs.org/javadoc/yamcs/org/yamcs/security/LdapAuthModule.html

9.4.2 YAML AuthModule

This AuthModule supports authentication and authorization of users via YAML files available directly in the Yamcs configuration folder.

9.4.2.1 Class Name

org.yamcs.security.YamlAuthModule[111]

9.4.2.2 Configuration Options

hasher (string)

Hasher class that can be used to verify if a password is correct without actually storing the password. When omitted, passwords in `users.yaml` should be defined in clear text. Possible values are:

- org.yamcs.security.PBKDF2PasswordHasher[112]

required (boolean)

When set to `true` the YAML AuthModule will veto the login process if it does not know the user. This may be of interest in situations where the YAML AuthModule does not authenticate the user, yet still some control is required via configuration files over which users can login. Default is `false`.

The YAML AuthModule reads further configuration from two additional YAML files: `users.yaml` and `roles.yaml`.

9.4.2.3 users.yaml

This file defines users, passwords and user roles. Note that Yamcs itself does not use roles, it is however used as a convenience in the YAML AuthModule to reduce the verbosity of user-specific privilege assignments.

```yaml
admin:
  password: somepassword
  superuser: true

someuser:
  displayName: Some User
  password: somepassword
  roles: [ Operator ]
```

The `password` key may be omitted if the YAML AuthModule is not used for authentication.

If you do use YAML AuthModule for authentication, consider hashing the passwords for better security. Password hashes can be obtained via the command line:

```
yamcsadmin password-hash
```

This command prompts for the password and outputs a randomly salted PBKDF2 hash. This output can be assigned to the `password` key, replacing the clear password.

9.4.2.4 roles.yaml

This file defines which privileges belong to which roles.

[111] https://yamcs.org/javadoc/yamcs/org/yamcs/security/YamlAuthModule.html

[112] https://yamcs.org/javadoc/yamcs/org/yamcs/security/PBKDF2PasswordHasher.html

```
Operator:
  ReadParameter: [".*"]
  WriteParameter: []
  ReadPacket: [".*"]
  Command: [".*"]
  CommandHistory: [".*"]
  System:
    - ControlProcessor
    - ModifyCommandHistory
    - ControlCommandQueue
    - GetMissionDatabase
    - ReadAlarms
    - ControlAlarms
    - ControlArchiving
```

This example specifies one role `Operator`. It also demonstrates the use of regular expressions to grant a set of object privileges.

System privileges must be defined under the key `System`. System privileges may not use regular expressions.

9.4.3 Kerberos AuthModule

This AuthModule supports password-based authentication of users via an external Kerberos server.

9.4.3.1 Class Name

org.yamcs.security.KerberosAuthModule[113]

9.4.3.2 Configuration Options

This module reads Kerberos configuration from the Kerberos system configuration file. This is usually available at `/etc/krb5.conf`. If you need to override this location, you have to set a system property at JVM level:

> -Djava.security.krb5.conf=/my/custom/krb5.conf

9.4.4 Remote User AuthModule

This AuthModule supports the login of users based on a provided HTTP header containing the username. Currently, it can only be used for API requests, and not for accessing the Yamcs web interface.

Warning: When using this module you must protect Yamcs against spoofing attacks.

9.4.4.1 Class Name

org.yamcs.security.RemoteUserAuthModule[114]

[113] https://yamcs.org/javadoc/yamcs/org/yamcs/security/KerberosAuthModule.html

[114] https://yamcs.org/javadoc/yamcs/org/yamcs/security/RemoteUserAuthModule.html

9.4.4.2 Configuration Options

header (string)

Name of the HTTP request header that indicates the remotely identified user.
Default: `X-REMOTE-USER`

9.4.5 SPNEGO AuthModule

This AuthModule supports Single Sign On authentication of users via SPNEGO. This is usually stacked together with the *Kerberos AuthModule* (page 119) module in case the single sign on does not work, or in case Yamcs is accessed from a non-web context.

9.4.5.1 Class Name

org.yamcs.security.SpnegoAuthModule[115]

9.4.5.2 Configuration Options

principal (string)

Required. Kerberos Service Principal of the HTTP service that matches the external address of Yamcs.
This should be in the format `HTTP/<host>.<domain>@<realm>`

keytab (string)

Required. Path to the keytab file matching the principal.

stripRealm (boolean)

Whether to strip the realm from the username (e.g. `user@<realm>` becomes just `user`).
Default: `true`.

This module reads Kerberos configuration from the Kerberos system configuration file. This is usually available at `/etc/krb5.conf`. If you need to override this location, you have to set a system property at JVM level:

```
-Djava.security.krb5.conf=/my/custom/krb5.conf
```

9.4.6 OpenID Connect AuthModule

New in version 5.0.0.

This AuthModule supports federated identity by redirecting web application users to the authorization (or consent) page of an OpenID Connect server. This allows for remote management of users and could be used to perform cross-domain Single Sign On with multiple other browser applications.

This AuthModule is used for authentication only. It does not directly support importing roles. But you could do so by extending this module.

[115] https://yamcs.org/javadoc/yamcs/org/yamcs/security/SpnegoAuthModule.html

9.4.6.1 Class Name

org.yamcs.security.OpenIDAuthModule[116]

9.4.6.2 Configuration Options

authorizationEndpoint (string)
: **Required.** The URL of the OpenID server page where to redirect users for authorization and/or consent.

tokenEndpoint (string)
: **Required.** The URL of the OpenID server page where OAuth2 tokens can be retrieved.

clientId (string)
: **Required.** An identifier that identifies your Yamcs server installation as a client against the Open ID server. This should be set up using the configuration tools of the Open ID server.

clientSecret (string)
: **Required.** The secret matching with the `clientId`.

scope (string)
: Space-separated scope to be used in authorization request. Default: `openid email profile`

attributes (map)
: Configure how claims are mapped to Yamcs attributes. If unset, Yamcs uses defaults that work out of the box against some common OpenID Connect providers.

9.4.6.2.1 Attributes sub-configuration

name (string or string[])
: The claim that matches with the account name. This is used internally by Yamcs to map the user to a single identity. If multiples are defined, they are tried in order. Default: `[preferred_username, nickname, email]`.

email (string or string[])
: The claim that matches with the email. If multiples are defined, they are tried in order. Default: `email`.

displayName (string or string[])
: The claim that matches with the display name. If multiples are defined, they are tried in order. Default: `name`.

Note to third-party developers

This AuthModule implements the conventions for server-side web applications. In other words: the `id_token` is retrieved and decoded on Yamcs server only. Before Yamcs can obtain the `id_token` it expects to be given some information by the integrating application.

The source code of the Yamcs web interface serves as the best reference. But generally it works like this:

1. The browser application retrieves OpenID Connect options on the `/auth` endpoint. This includes the `client_id`, the `authorizationEndpoint` and the `scope`. Other configuration options are reserved for server use.
2. The browser application uses the `authorizationEndpoint` to redirect the browser to a login or consent page of the upstream OIDC server. The following is an example:

[116] https://yamcs.org/javadoc/yamcs/org/yamcs/security/OpenIDAuthModule.html

```
window.location.href = "https://oidc.example.com" +
        "?client_id=encodeURIComponent(CLIENT_ID)" +
        "&state=encodeURIComponent(STATE)" +
        "&response_mode=query" +
        "&response_type=code" +
        "&scope=openid+email+profile" +
        "&redirect_uri=encodeURIComponent(REDIRECT_URI)";
```

`STATE` can be anything, and is typically used for encoding information about the original request such that when the authentication is done, the user is redirected back to the original attempted route.

`REDIRECT_URI` is the path where OIDC will redirect back the user after the login or consent is confirmed.

3. When OIDC redirects the user's browser back to REDIRECT_URI, extract the `code` and `state` from the query params.

4. Use this upstream `code` to make an encoded string like this:

   ```
   var codeForYamcs = "oidc " + JWT;
   ```

   Here, JWT represent a JSON Web Token that stringifies a payload containing at least these properties:

   ```
   {
     "redirect_uri": REDIRECT_URI,
     "code": UPSTREAM_CODE,
   }
   ```

5. The string value of the variable `codeForYamcs` can be used against the Yamcs `/auth` endpoint using `grant_type=authorization_code` for converting it to a standard Yamcs-level access token.

   In the background what happens is that Yamcs will use the upstream code and exchange it against OIDC for an `id_token` which tells Yamcs what the username, email and display name are for the authenticated user. The `redirect_uri` property is not actually used anymore, but most OIDC servers will check on this being specified and matching the original `redirect_uri` used for obtaining the upstream code.

The security subsystem is modular by design and allows combining different AuthModules together. This allows for scenarios where for example you want to authenticate via LDAP, but determine privileges via YAML files.

The default set of AuthModules include:

LDAP AuthModule **(page 117)**
Attempts to bind to LDAP with the provided credentials. Also capable of reading privileges for the user.

YAML AuthModule **(page 118)**
Reads Yaml files to verify the credentials of the user, or assign privileges.

Kerberos AuthModule **(page 119)**
Supports authenticating against a Kerberos server.

Remote User AuthModule **(page 119)**
Supports authentication based on a custom HTTP header.

SPNEGO AuthModule **(page 120)**
Supports authenticating against a Kerberos server using Single Sign On from a web context.

OpenID Connect AuthModule **(page 120)**
Supports authenticating against an OpenID Connect server.

AuthModules have an order. When a login attempt is made, AuthModules are iterated a first time in this order. Each AuthModule is asked if it can authenticate with the provided credentials. The first matching AuthModule contributes the user principal. A second iteration is done to then contribute privileges to the identified user. During both iterations, AuthModules reserve the right to halt the global login process for any reason.

Some AuthModules are only useful for specific flows. For example OpenID Connect (which in a nutshell redirects to an external login form) would need to be accompanied with other AuthModules in case not all target clients are browser-based.

9.5 Configuration

The security system is configured in the file `etc/security.yaml`. Example:

```yaml
authModules:
  - class: org.yamcs.security.LdapAuthModule
    args:
       ...
```

This requires that all login attempts are validated against an external LDAP server.

These options are supported:

authModules (list of maps)

List of AuthModules that participate in the login process. Each AuthModule may support custom configuration options which can be defined under the `args` key. If empty only the internal Yamcs directory is used as a source of users and roles.

blockUnknownUsers (boolean)

Use this if you need fine control over who can access Yamcs. Successful login attempts from users that were not yet known by Yamcs will be blocked by default. A privileged user may unblock them. The typical use case is when Yamcs uses an external identity provider that allows more users than really should be allowed access to Yamcs.

Default: false

enabled (boolean)

Control whether authentication is enforced.

Default: `true` if `security.yaml` is present, `false` otherwise.

guest (map)

Overrides the user properties of the guest user. This user is used for all access when authentication is not being enforced.

10. Web Interface

Yamcs includes a web interface which provides quick access and control over many of its features. The web interface runs on port 8090 and integrates with the security system of Yamcs.

All pages are aware of the privileges of the logged in user and will hide user interface elements that the user has no permission for.

Most pages (the homepage excluding) show data specific to a particular Yamcs instance. The current instance is always indicated in the top bar. To switch to a different location either return to the homepage, or use the quick-switch dialog in the top bar. When switching instances the user is always redirected to the default page for that instance.

10.1 Configuration

10.1.1 Configuration Options

`tag` (string)
Short descriptor string of this Yamcs server. If present this is shown in the top bar. The primary motivation for this option is to be able to distinguish between multiple Yamcs servers in distributed deployments.

`extraHeaderHTML` (string)
Additional HTML to be included at the end of the <head></head> section.

`displayBucket` (string)
Bucket where to find display resources.

Default: `displays`

`displayFolderPerInstance` (boolean)
Controls whether displays are specific to an instance.

Avoid using this option, there are plans to deprecate it once instances can choose a specific bucket in their configuration.

Default: `false`

`stackBucket` (string)
Bucket where to find stacks.

Default: `stacks`

`stackFolderPerInstance` (boolean)
Controls whether stacks are specific to an instance.

Avoid using this option, there are plans to deprecate it once instances can choose a specific bucket in their configuration.

Default: `false`

`staticRoot` (string)
Filesystem path where to locate the web files for the Yamcs Web Interface (*.js, *.css, *.html, …). If not specified, Yamcs will search the classpath for these resources (preferred).

It should only be necessary to use this option when doing development work on the Yamcs Web Interface. It allows to run npm in watch mode for a save-and-refresh development cycle.

`twoStageCommanding` (boolean)
Controls whether to protect commanding from from accidental clicks. If true issuing a command will require two clicks at least (arm-and-issue).

This feature is primarily intended for an operational setting.

Default: `false`

`collapseInitializedArguments` (boolean)
Controls the display of argument fields of a command form. If true, arguments with an initial value are collapsed by default.

Default: `true`

`commandExports` (boolean)
Controls whether the button to generate an offline command report is shown.

Default: `true`

`logoutRedirectUrl` (string)
The URL to redirect to after logging out of Yamcs. If unset, users are redirected to the built-in login page.

`dass` (boolean)
Controls whether functionalities related to DaSS gap retrieval are shown.

This function requires proprietary services that are not available in the open-source Yamcs distribution, and shall be removed whenever technically possible.

Default: `false`

`tc` (boolean)
Controls whether commmanding functionalities are shown.

Default: `true`

`tmArchive` (boolean)
Controls whether functionalities related to TM archiving are shown.

Default: `true`

`events` (map)
Configure event-related properties.

See “Events sub-configuration” section below.

Events sub-configuration

`extraColumns` (list of maps)
Additional event columns specific to a deployment.

See “Extra Columns sub-configuration” section below.

Extra Columns sub-configuration

`id` (string)
Required. Identifier of this column. This corresponds with a key in the extra map of an event.

`label` (string)
Required. Descriptive name of this column.

`visible` (boolean)
Whether this column is visible by default.

Default: `true`

`alwaysVisible` (boolean)
Whether this column always visible.

If false, the user may choose to hide it.

Default: `false`

`after` (string)
Required. Identifier of the column after which to position this column.

One of `severity`, `gentime`, `message`, `source`, `type`, `rectime` or `seqNumber`

10.2 Links

Shows a live view of the data links for this instance. Link can be managed directly from this page.

10.3 Telemetry

The Telemetry group within the Yamcs web interface provides access to monitoring-related pages.

10.3.1 Parameters

This page lists all parameters. Each parameter can be accessed individually to see the latest or archived values. Numeric parameters can also be charted.

10.3.2 Displays

Shows the displays or display resources that are known by Yamcs Server for the selected instance. The displays in this view are presented in a file browser with the usual operations to rename, move or create. Clicking on a display file opens the display. If there is incoming telemetry this will be received by the opened display file.

Note that only some display types are supported by the Yamcs web interface. The following provides an overview of the current state:

Extension	Display Type	View	Edit
`opi`	Yamcs Studio displays	Basics	No plans to support (use Yamcs studio)
`par`	Parameter tables	Full support	Full support

In addition there is file preview support for the following display resources:

Extension	Resource Type	View	Edit
png, gif, bmp, jpg, jpeg	Image	Full support	No plans to support
js	Script file	Full support	Planned

Any other file is displayed in a basic text viewer.

10.4 Events

This section provides a view on Yamcs events. By default only the latest events within the last hour get shown. The view offers ample filter options to change which events are shown. The table is paged to prevent overloading the browser. If you like to see beyond the current page, you can click the button 'Load More' at the bottom of the view. Alternatively you can choose to click the 'Download Data' button at the top right. This will trigger a download of the events in CSV format. The download will apply the same filter as what is shown in the view.

The Events table can also monitor incoming events on the current processor. Do so by clicking the play button in the top toolbar. You may stop the live streaming at any time by clicking the pause button.

The Events table has a severity filter. This filter allows defining the **minimum** severity of the event. Events that are more severe than the selected severity will also be shown. By default the severity filter is set to the lowest severity, `Info`, which means that all events will be shown.

With the right privilege, it is possible to manually post an event. You can enter an arbitrary message and assign a severity. The time of the event will by default be set to the current time, but you can override this if preferred. The source of an event created this way will automatically be set to `User` and will contain a `user` attribute indicating your username.

10.5 Alarms

Shows an overview of the current alarms. Alarms indicate parameters that are out of limits.

10.6 Commanding

10.6.1 Send a Command

Prepare and send a single command.

10.6.2 Command History

Shows the latest issued commands.

10.7 MDB

The MDB module within the Yamcs web interface provides a set of views on the Mission Database.

The MDB Module is always visited for a specific Yamcs instance. The MDB for an instance aggregates the content of the entire MDB loader tree for that instance.

10.7.1 Parameters

The Parameters view shows a filterable list of all parameters inside the MDB. If you are searching for a specific parameter but don't remember the space system this views can help find it quickly.

You can navigate to the detail page of any parameter to see a quick look at its definition, and to see the current realtime value. If the parameter has numeric values, its data can also be rendered on a chart. This chart is updated in realtime. Finally the detail page of a parameter also has a view that allows looking at the exact data points that have been received in a particular time range. This information is presented in a paged view. There is a download option available for downloading data points of the selected time range as a CSV file for offline analysis.

If the parameter is a software parameter, its value can be set via a button in the toolbar.

10.7.2 Containers

The Containers view shows a filterable list of all containers inside the MDB. The detail page allows seeing the parameter or container entries for this container and offers navigation links for quick access.

10.7.3 Commands

The Commands view shows a filterable list of all commands inside the MDB. This also includes abstract commands. Non-abstract commands can be issued directly from the detail page of that command. This opens a dynamic dialog window where you can override default arguments and enter missing arguments.

10.7.4 Algorithms

The Algorithms view shows a filterable list of all algorithms inside the MDB. This detail page provides a quick navigation list of all input and output parameters and shows the script for this algorithm.

10.8 Archive

10.8.1 Overview

This view allows inspecting the content of the TM Archive, as well as retrieving data as packets. Data is grouped by packet name in bands. For each band, index blocks indicate the presence of data at a particular time range. Note that a single index block does not necessarily mean that there was no gap in the data. When zooming in, more gaps may appear.

The view can be panned by grabbing the canvas. For long distances you can jump to a specific location via the `Jump to...` button.

This view shows the current mission time with a vertical locator.

Note: While the now locator follows mission time, the rendered blocks do not follow realtime. You can force a refresh by panning the canvas or refreshing your browser window.

In the top toolbar there are a few actions that only become active once you make a horizontal range selection. To make such a selection you can start a selection on the timescale band. Alternatively you may also select a range by simply clicking an index block. Selecting a range allows you to start a replay for that range, or to download raw packet data.

10.8.2 Tables

Shows the archive tables for this instance. Each table has a detail page that shows details about its structure and SQL options and that provides a quick view at the raw data records.

10.8.3 Streams

Shows the streams for this instance. Each stream has a detail page that shows details about its SQL definition.

10.9 Admin Area

The Admin Area within the Yamcs web interface provides a set of administrative views on Yamcs.

10.9.1 Services

Shows the services for this instance. The lifecycle of these services can be managed directly from this page.

11. Programs

11.1 yamcsadmin

NAME

yamcsadmin - Tool for local Yamcs administration

SYNOPSIS

```
yamcsadmin [--etc-dir DIR] COMMAND
```

OPTIONS

--log LEVEL
Level of verbosity. From 0 (off) to 5 (all). Default: 2.

--etc-dir DIR
Override default Yamcs configuration directory.

--data-dir DIR
Override default Yamcs data directory.

-h, --help
Show usage.

-v, --version
Print version information and quit.

COMMANDS

backup **(page 131)**
Perform and restore backups

confcheck **(page 134)**
Check Yamcs configuration

mdb **(page 134)**
Provides MDB information

parchive **(page 135)**
Parameter Archive operations

password-hash **(page 135)**
Generate password hash for use in users.yaml

rocksdb **(page 136)**
Provides low-level RocksDB data operations

users **(page 137)**
User operations

11.1.1 yamcsadmin backup

NAME

yamcsadmin backup - Perform and restore backups

SYNOPSIS

```
yamcsadmin backup COMMAND
```

COMMANDS

create **(page 132)**
Create a new backup.

delete **(page 132)**
Delete a backup.

list **(page 133)**
List the existing backups.

purge **(page 133)**
Purge old backups.

restore **(page 133)**
Restore a backup.

11.1.1.1 yamcsadmin backup create

NAME

yamcsadmin backup create - Create a new backup

SYNOPSIS

```
yamcsadmin backup create --backup-dir DIR [--data-dir DIR]
                         [--url HOST:PORT] TABLESPACE
```

DESCRIPTION

This subcommand allows to create either a hot or a cold backup of a Yamcs tablespace. For cold backups, specify the `--data-dir` property, for hot backups specify the `--host` property.

POSITIONAL ARGUMENTS

`TABLESPACE`
The name of the tablespace to backup.

OPTIONS

`--backup-dir` `DIR`
Target directory containing backups. This directory is automatically created if it does not exist prior to taking the backup.

`--data-dir` `DIR`
Path to a Yamcs data directory. This must be specified when performing a cold backup.

`--host` `HOST:PORT`
Perform a hot backup. This allows to take a consistent backup while Yamcs is running. Backup are currently triggered using a JMX operation.

11.1.1.2 yamcsamdin backup delete

NAME

yamcsadmin backup delete - Delete a backup

SYNOPSIS

```
yamcsadmin backup delete --backup-dir DIR ID ...
```

POSITIONAL ARGUMENTS

`ID ...`
Backup IDs to delete.

OPTIONS

`--backup-dir DIR`
Directory containing backups.

11.1.1.3 yamcsadmin backup list

NAME

yamcsadmin backup list - List the existing backups

SYNOPSIS

```
yamcsadmin backup list --backup-dir DIR
```

OPTIONS

`--backup-dir DIR`
Directory containing backups.

11.1.1.4 yamcsamdin backup purge

NAME

yamcsadmin backup purge - Purge old backups

SYNOPSIS

```
yamcsadmin backup purge --backup-dir DIR --keep N
```

OPTIONS

`--backup-dir DIR`
Directory containing backups.

`--keep N`
The number of backups to keep

11.1.1.5 yamcsadmin backup restore

NAME

yamcsadmin backup restore - Restore a backup

SYNOPSIS

```
yamcsadmin backup restore --backup-dir DIR --restore-dir DIR [ID]
```

DECRIPTION

Note that backups can only be restored when Yamcs is not running.

POSITIONAL ARGUMENTS

`ID`
Backup ID. If unspecified this defaults to the last backup.

OPTIONS

`--backup-dir DIR`
Directory containg backups.

--restore-dir DIR

Directory where to restore the backup.

11.1.2 yamcsadmin confcheck

NAME

yamcsadmin confcheck - Check Yamcs configuration

SYNOPSIS

```
yamcsadmin confcheck
```

11.1.3 yamcsadmin mdb

NAME

yamcsadmin mdb - Provides MDB information

SYNOPSIS

```
yamcsadmin mdb COMMAND
```

COMMANDS

print

Print MDB content

verify

Verify that the MDB can be loaded

11.1.3.1 yamcsadmin mdb print

NAME

yamcsadmin mdb print - Print MDB content

SYNOPSIS

```
yamcsadmin mdb print INSTANCE
```

POSITIONAL ARGUMENTS

INSTANCE

Instance name.

11.1.3.2 yamcsadmin mdb verify

NAME

yamcsadmin mdb verify - Verify that the MDB can be loaded

SYNOPSIS

```
yamcsadmin mdb verify INSTANCE
```

POSITIONAL ARGUMENTS

INSTANCE

Instance name.

11.1.4 yamcsadmin parchive

NAME

yamcsadmin parchive - Parameter Archive operations

SYNOPSIS

```
yamcsadmin parchive --instance INSTANCE COMMAND
```

OPTIONS

--instance INSTANCE
Yamcs instance.

COMMANDS

print-pid **(page 135)**
Print parameter name to parameter id mapping.

print-pgid **(page 135)**
Print parameter group compositions.

11.1.4.1 yamcsadmin parchive print-pid

NAME

yamcsadmin parchive print-pid - Print parameter name to parameter id mapping

SYNOPSIS

```
yamcsadmin parchive print-pid
```

11.1.4.2 yamcsadmin parchive print-pgid

NAME

yamcsadmin parchive print-pgid - Print parameter group compositions

SYNOPSIS

```
yamcsadmin parchive print-pgid
```

11.1.5 yamcsadmin password-hash

NAME

yamcsadmin password-hash - Generate password hash for use in users.yaml

SYNOPSIS

```
yamcsadmin password-hash
```

DESCRIPTION

Prompts to enter and confirm a password, and generates a randomly salted PBKDF2 hash of this password. This hash may be used in users.yaml instead of the actual password, and allows verifying user passwords without storing them.

The command may be used in non-interactive mode by setting the password with the environment variable `YAMCSADMIN_PASSWORD`

11.1.6 yamcsadmin rocksdb

NAME

yamcsadmin rocksdb - Provides low-level RocksDB data operations

SYNOPSIS

```
yamcsadmin rocksdb COMMAND
```

COMMANDS

***compact* (page 136)**
Compact RocksDB database

***bench* (page 136)**
Benchmark RocksDB storage engine

11.1.6.1 yamcsadmin rocksdb compact

NAME

yamcsadmin rocksdb compact - Compact RocksDB database

SYNOPSIS

```
yamcsadmin rocksdb compact [--dbDir DIR] [--sizeMB SIZE]
```

OPTIONS

`--dbDir` `DIR`
Database directory.

`--sizeMB` `SIZE`
Target size of each SST file in MB (default is 256 MB).

11.1.6.2 yamcsadmin rocksdb bench

NAME

yamcsadmin rocksdb bench - Benchmark RocksDB storage engine

SYNOPSIS

```
yamcsadmin rocksdb bench [--dbDir DIR] [--baseTime TIME]
    [--count COUNT] [--duration HOURS]
```

OPTIONS

`--dbDir` `DIR`
Directory where the database will be created. A “rocksbench” archive instance will be created in this directory

`--baseTime` `TIME`
Start inserting data with this time. Default: 2017-01-01T00:00:00

`--count` `COUNT`
The partition counts for the 5 frequencies: [10/sec, 1/sec, 1/10sec, 1/60sec and 1/hour]. It has to be specified as a string (use quotes).

`--duration` `HOURS`
The duration in hours of the simulated data. Default: 24

DESCRIPTION

The benchmark consists of a table load and a few selects. The table is loaded with telemetry packets received at frequencies of [10/sec, 1/sec, 1/10sec, 1/60sec and 1/hour]. The table will be identical to the tm table and will contain a histogram on pname (= packet name). It is possible to specify how many partitions (i.e. how many different pnames) to be loaded for each frequency and the time duration of the data.

11.1.7 yamcsadmin users

NAME

yamcsadmin users - User operations

SYNOPSIS

```
yamcsadmin users COMMAND
```

COMMANDS

add-role **(page 137)**
Add a role to a user.

create **(page 138)**
Check a user's password.

create **(page 138)**
Create new user.

delete **(page 139)**
Delete a user.

describe **(page 139)**
Describe user details.

list **(page 139)**
List users.

remove-role **(page 139)**
Remove a role from a user.

reset-password **(page 140)**
Reset a user's password.

update **(page 140)**
Update user details.

11.1.7.1 yamcsadmin users add-role

NAME

yamcsadmin users add-role - Add a role to a user

SYNOPSIS

```
yamcsadmin users add-role USERNAME --role ROLE
```

DESCRIPTION

Adds a role to a user.

POSITIONAL ARGUMENTS

`USERNAME`
The name of the user.

OPTIONS

--role ROLE
: Role to be added.

11.1.7.2 yamcsadmin users check-password

NAME

yamcsadmin users check-password - Check a user's password

SYNOPSIS

```
yamcsadmin users check-password USERNAME
```

DESCRIPTION

Prompts to enter the user's current password. The command will print if the provided password is correct or not.

The command may be used in non-interactive mode by setting the password with the environment variable `YAMCSADMIN_PASSWORD`.

POSITIONAL ARGUMENTS

USERNAME
: The name of the user.

11.1.7.3 yamcsadmin users create

NAME

yamcsadmin users create - Create a new user

SYNOPSIS

```
yamcsadmin users create [--email EMAIL] [--display-name NAME]
    [--inactive] [--superuser] [--no-password] USERNAME
```

DESCRIPTION

Create a new Yamcs user. This prompts for a password.

The command may be used in non-interactive mode by setting the password with the environment variable `YAMCSADMIN_PASSWORD`, or using the option `--no-password`.

POSITIONAL ARGUMENTS

USERNAME
: The name of the new user.

OPTIONS

--display-name NAME
: Displayed name of the user.

--email EMAIL
: User email.

--inactive
: Add this flag to prevent Yamcs from activating the account.

--superuser
: Add this flag to grant this user superuser privileges.

--no-password
: Add this flag to indicate that this user should not have a password. This will also bypass the password prompt.

11.1.7.4 yamcsadmin users delete

NAME

yamcsadmin users delete - Delete a user

SYNOPSIS

```
yamcsadmin users delete USERNAME
```

POSITIONAL ARGUMENTS

USERNAME
The name of the user.

11.1.7.5 yamcsadmin users describe

NAME

yamcsadmin users describe - Describe user details

SYNOPSIS

```
yamcsadmin users describe USERNAME
```

POSITIONAL ARGUMENTS

USERNAME
The name of the user.

11.1.7.6 yamcsadmin users list

NAME

yamcsadmin users list - List users

SYNOPSIS

```
yamcsadmin users list
```

11.1.7.7 yamcsadmin users remove-role

NAME

yamcsadmin users remove-role - Remove a role from a user

SYNOPSIS

```
yamcsadmin users remove-role USERNAME --role ROLE
```

DESCRIPTION

Remove a role from a user.

POSITIONAL ARGUMENTS

USERNAME
The name of the user.

OPTIONS

--role ROLE
Role to be removed.

11.1.7.8 yamcsadmin users reset-password

NAME

yamcsadmin users reset-password - Reset a user's password

SYNOPSIS

`yamcsadmin users reset-password USERNAME`

DESCRIPTION

Prompts to enter and confirm a new user password.

The command may be used in non-interactive mode by setting the password with the environment variable `YAMCSADMIN_PASSWORD`.

POSITIONAL ARGUMENTS

`USERNAME`
The name of the user.

11.1.7.9 yamcsadmin users update

NAME

yamcsadmin users update - Update a user

SYNOPSIS

```
yamcsadmin users update [--active true | false]
    [--display-name NAME] [--email EMAIL]
    [--superuser true | false] USERNAME
```

POSITIONAL ARGUMENTS

`USERNAME`
The name of the user.

OPTIONS

`--display-name` `NAME`
Displayed name of the user.

`--email` `EMAIL`
User email.

`--active` `true | false`
Whether the user account is active or not.

`--superuser`
Whether the user has superuser privileges.

11.2 yamcsd

NAME

yamcsd - Yamcs Server

SYNOPSIS

```
yamcsd [--version] [--help] [--check] [--log LEVEL] [--log-config FILE]
       [--no-color] [--no-stream-redirect] [--etc-dir DIR]
       [--data-dir DIR] [--netty-leak-detection LEVEL]
```

DESCRIPTION

yamcsd is a shell wrapper that launches a JVM running the YamcsServer main program.

OPTIONS

--log LEVEL
Level of verbosity. From 0 (off) to 4 (all). Default: 2. This option only affects console logging, not file logging. For high verbosity levels, this option should be combined with the option `--log-config` to reduce the amount of output to only selected individual loggers.

--log-config FILE
Finetune the log level of individual loggers. This option only affects console logging, not file logging. An example is given below. When this option is not specified, all loggers are active.

--no-color
Add this flag to disable ANSI color codes used in console logging.

--no-stream-redirect
Add this flag to prevent Yamcs from redirecting stdout/stderr output via the logging system.

--etc-dir DIR
Path to config directory. This defaults to the etc directory relative to the working directory.

--data-dir DIR
Path to data directory. When unspecified the location is read from the `yamcs.yaml` configuration file.

--check
Run syntax tests on configuration files and quit.

--netty-leak-detection LEVEL
Level of leak detection used by the Netty library. Leak detection is disabled by default as it has a negative impact on performance. The available levels are:

DISABLED
Disables leak detection (default)

SIMPLE
Samples 1% of all Netty resources and reports when a leak is detected. Small overhead, but difficult to tell what caused the leak.

ADVANCED
Samples 1% of all Netty resources and reports when a leak is detected and where the object was recently accessed. High overhead.

PARANOID
Tracks all Netty resources and reports when a leak is detected and where the object was recently accessed. Very high overhead.

Note that leak detection triggers only upon a GC.

-v, --version
Print version information and quit.

-h, --help
Show usage.

LOG CONFIG EXAMPLE

The file specified with the option `--log-config` must be in properties format, where keys represent a logger, and values represent the verbosity level of that logger. Unmentioned loggers are considered to be off (level = 0). Example:

```
# Levels:
# 0 = off
# 1 = warnings and errors
# 2 = info
# 3 = debug
# 4 = trace
```

(continues on next page)

(continued from previous page)

```
org.yamcs = 3
org.yamcs.http = 1
com.example.myproject = 4
```

Note that the effective log level of any specified logger is always ceiled to that of the `--log` option.

11.3 Systemd Unit File

Yamcs package installations include a systemd unit file for starting and stopping Yamcs as a service.

The unit file is located at `/usr/lib/systemd/system/yamcs.service`.

You should not modify this file directly, but instead use standard systemd mechanisms to customize unit files. See the instructions for your operating system.

Usage:

```
systemctl start|stop|restart|status yamcs
```

systemctl accepts these commands:

`start`
Starts Yamcs.

`stop`
Stops the Yamcs process and any other processes it may have launched.

`restart`
Stops Yamcs if it is running, then starts it again.

`status`
Checks if Yamcs is currently running. This will only detect a Yamcs runtime that has been started via systemd.

If you would like Yamcs to start automatically on boot, run:

```
systemctl enable yamcs
```

If you want to revert Yamcs starting automatically, run:

```
systemctl disable yamcs
```

12. Configuration Sections

Some of the standard configuration files can be extended with custom configuration options. This is called a configuration section. Sections are represented by a top-level identifier and are scoped to a type of configuration file.

A Yamcs plugin is automatically associated with a configuration section named after the plugin identifier.

For example, the `yamcs-web` module is packaged as a Yamcs plugin, and accepts configuration options read from the `yamcs-web` section of the main `yamcs.yaml`:

```
public class WebPlugin implements Plugin {

    public Spec getSpec() {
        Spec spec = new Spec();
        // ...
        return spec;
    }

    @Override
    public void onLoad(YConfiguration config) throws PluginException {
        // Use the actual configuration
    }
}
```

Here the org.yamcs.Spec[117] object is a helper class that allows defining how to validate your plugin configuration. Yamcs will take care of the actual validation step, and if all went well the `onLoad` hook should trigger. This is a good place to access the runtime configuration model, and retrieve your custom options.

If you have custom components that want to access this configuration, one possible way is to provide accessors on your plugin class, and then to retrieve the singleton instance of your plugin class:

```
PluginManager pluginManager = YamcsServer.getServer().getPluginManager();
MyPlugin plugin = pluginManager.getPlugin(MyPlugin.class);
// ...
```

Instance-specific configuration

Besides global plugin configuration options in `yamcs.yaml`, you may also want to add instance-specific configuration options. These would be considered when validating any `yamcs.[instance].yaml` file:

```
YamcsServer yamcs = YamcsServer.getServer();
yamcs.addConfigurationSection(ConfigScope.YAMCS_INSTANCE, "my-section", spec);
```

[117] https://yamcs.org/javadoc/yamcs/org/yamcs/Spec.html

13. Command Options

New in version 5.0.0.

Yamcs supports a programmatic API for activating custom command options. When commands are issued with custom options, these can be interpreted by you own code, typically a TC data link.

Custom command options do not impact the encoding of telecommand packets, rather they are used for passing other instructions, such as at-runtime overriding of link properties.

Custom command options are added system-wide. Registered command options are available in all official clients wherever a command can be configured for sending.

Command options are automatically saved as attributes in Command History, and will also be received by all command/acknowledgment listeners.

13.1 Registration

Command options must be registered against org.yamcs.YamcsServer[118]. It is not possible to send custom options without the option being registered.

```
// Statically retrieve the current Yamcs server object.
YamcsServer yamcs = YamcsServer.getServer();

CommandOption option = new CommandOption(
    "cop1Bypass",  // System-wide unique identifier. Also stored in cmdhist.
    "COP-1 Bypass", // Verbose name for display in UI clients.
    CommandOptionType.BOOLEAN, // The expected type for hinting UI clients.
);

yamcs.addCommandOption(option);
```

A registration can only be done once, or else `addCommandOption()` will throw an exception. One way of doing so is to put this registration in the static initializer of the components that uses this option (e.g. a command link). Then the command option will only be loaded (and once only) when at least one such link is running.

An alternative method that avoids the use of static initializers, is to implement org.yamcs.Plugin[119], and then put the registration in the `onLoad` lifecycle hook. This hook is called once-only when the server is starting up.

```
public class MyPlugin implements Plugin {

    public static final CommandOption MY_OPTION = ...;

    public void onLoad(YConfiguration config) { // Called on start-up
        YamcsServer yamcs = YamcsServer.getServer();
        yamcs.addCommandOption(MY_OPTION);
    }
}
```

[118] https://yamcs.org/javadoc/yamcs/org/yamcs/YamcsServer.html
[119] https://yamcs.org/javadoc/yamcs/org/yamcs/Plugin.html

Note: Plugins must be packaged in a specific manner, before Yamcs can actually find and load them. This is documented separately.

13.2 Types

There is support for these three types: `BOOLEAN`, `NUMBER` and `STRING`. These types are only a hint for use by UI clients. For example, the Yamcs web interface will use these types to determine which UI controls to render in a dynamic form, and how to encode the values for persisting in Command History. The HTTP API will not check which Value[120] types are used. Submitted values are pushed end-to-end in a type-preserving manner.

The effective Value[121] type is intentionally loose, and depends on the client. The Yamcs web interface for example, will use doube for submitting the value of any `NUMBER` options.

13.3 Permissions

The use of any command option is subject to the system privilege *CommandOptions*. This is because our known use cases use this feature only for overriding advanced options requiring elevated privileges.

[120] https://yamcs.org/javadoc/yamcs/org/yamcs/protobuf/Value.html

[121] https://yamcs.org/javadoc/yamcs/org/yamcs/protobuf/Value.html

14. Yamcs Plugin Format

Yamcs has a simple plugin system that facilitates hooking into internals. The main advantages is that it allows to trigger custom code on server-start, which makes it an ideal place for programmatic customizations.

For example, you could use a plugin to dynamically add services without even needing to write them in YAML. Or you could use a plugin to read and validate some custom configuration file that is shared by multiple of your components. Or maybe you want to add your own custom HTTP and WebSocket calls to the API.

The following is a detailed specification of how Yamcs plugins should be packaged. If you want the short instructions, just implement org.yamcs.Plugin[122] and add this execution to the `pom.xml` of your Yamcs Maven project. Then everything will be derived automatically:

```xml
<plugin>
  <groupId>org.yamcs</groupId>
  <artifactId>yamcs-maven-plugin</artifactId>
  <!--version>...</version-->
  <executions>
    <execution>
      <goals>
        <goal>detect</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

14.1 Main configuration file

Yamcs plugins should be packaged inside regular jar files. You can have as many plugins inside a jar as you want. Your jar file must contain the following file in its classpath:

> /META-INF/services/org.yamcs.Plugin

The content of this file must list the class names of all the plugins in your jar (one on each line). So for instance if you want to register your plugin `com.example.MyPlugin`, then the contents of the file `org.yamcs.Plugin` must be simply:

```
com.example.MyPlugin
```

With this setup, Yamcs will find your plugin and hook it into its lifecycle.

14.2 Plugin metadata

In addition to the file `/META-INF/services/org.yamcs.Plugin`, you must also add the following file to your classpath:

> /META-INF/yamcs/com.example.MyPlugin/plugin.properties

[122] https://yamcs.org/javadoc/yamcs/org/yamcs/Plugin.html

Replace `com.example.MyPlugin` with the class name of your own plugin. The file `plugin.properties` supports the following key value pairs:

```
# REQUIRED. A short identifier for your plugin
name=my-plugin

# REQUIRED. A version number for your plugin
version=1.0.0

# Optional: freeform description (no markup)
description=Example

# Optional: your organization name
organization=Example

# Optional: your organization's URL
organizationUrl=https://example.com

# Optional: when your plugin package was generated (ISO 8601)
generated=
```

All these properties are used by Yamcs as metadata for potential integration in APIs and UIs.

Index