Space Applications Services

Yamcs Server Manual

YAMCS-SA-MA-001

February, 21st
2019

www.yamcs.org

Yamcs Server Manual

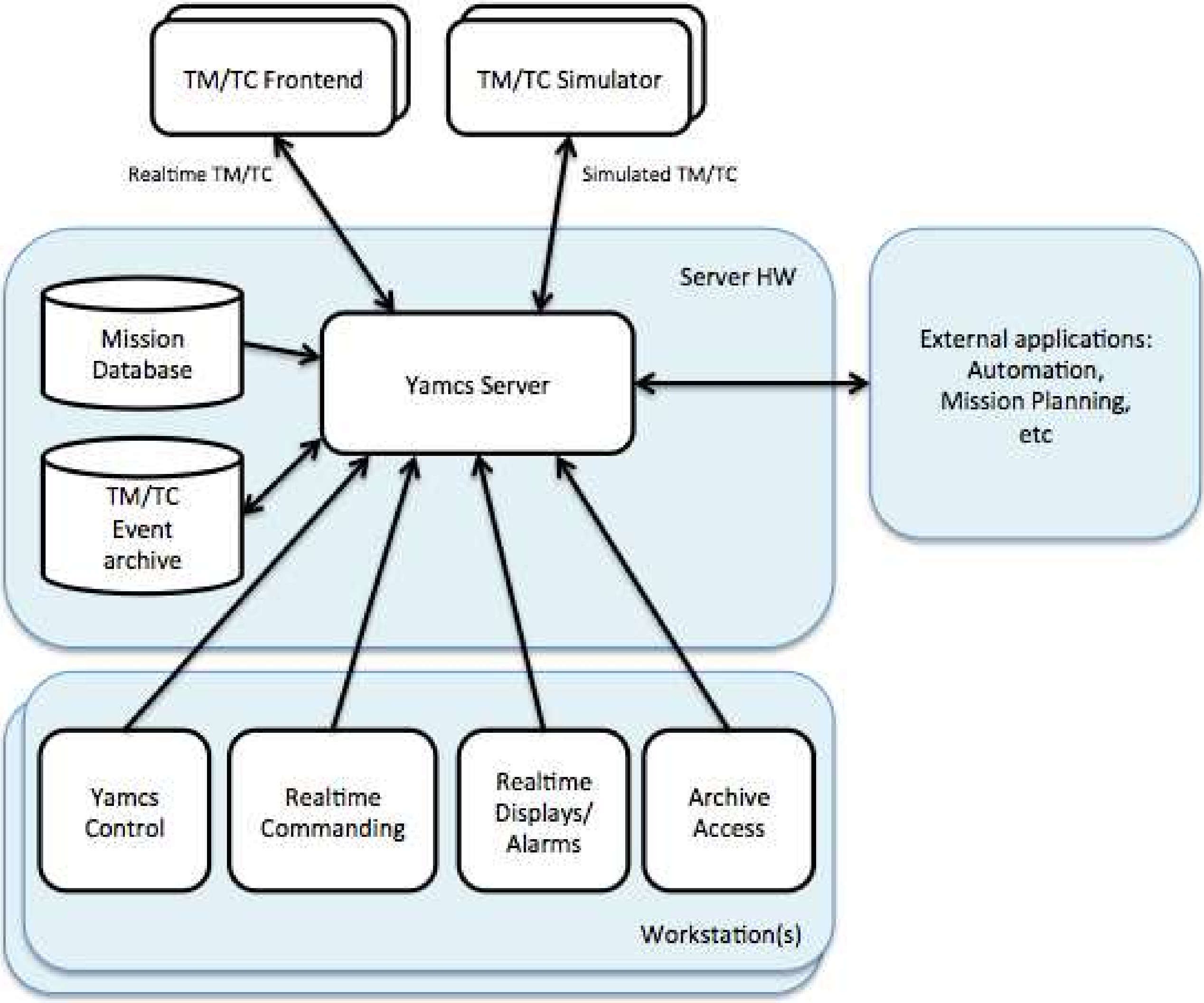
YAMCS-SA-MA-001

This version was published February, 21st 2019
A later version of this document may be available at www.yamcs.org

Table of Contents

Chapter 1. Introduction

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. The preferred target platform is linux x64, but Yamcs can also be made to run on Mac OS X and Windows.

While Yamcs binaries are typically used to run the Yamcs Server, they can also be useful as a library for reusable tasks such as processing of packet telemetry according to XTCE definitions. This manual describes how to use Yamcs as a server. If you want to use it for other purposes, please contact the developers at Yamcs mailing list.

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) but not yet aggregate types (aka structs in C language) - for example to represent an (x,y,z) position. Yamcs does preserve the association between parameters coming in the same group, which helps alleviating the problem of missing aggregates.

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 the XTCE standard, 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 regularely, 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 throguh 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. Fundamentals

Mission Database

The Mission Database is a dictionary containing the description of all the telemetry packets, parameters and commands. Yamcs implements concepts from the XTCE standard (XML Telemetric & Command Exchange). Although this standard specifies how to describe a mission database in XML, it also prescribes a series of building blocks for such a database: SpaceSystens, containers, parameters, commands, algorithms, etc.

In Yamcs we do not have as a primary goal the compliance with the XML representation (although Yamcs does have an XTCE XML loader) but more to the structure and the concepts of an XTCE database.
In fact the primary representation for the mission database in Yamcs is an excel spreadsheet because we noticed that many operators prefer to work with such tool. It is however possible to create database loaders for different formats and it has been done many times in order to be interoperable with other tools (integration is key aspect of Yamcs).

Note: XTCE is a standard created by many space agencies and thus it contains quite a wide range of features. The standard is also evolving with version 1.2 (not yet public) having many changes from version 1.1. Yamcs does not implement completely the standard, thus we cannot state compliance with a particular version.

Instances

The Yamcs instances provide means for one Yamcs server to monitor/control different payloads or sattelites or version of the payloads or satellites at the same time. Each instance has a name and a directory where all data from that instance is stored, as well as a specific Mission Database used to process data for that instance. Therefore, each time the Mission Database changes (e.g. due to an on-board software upgrade), a new instance has to be created. One strategy to deal with long duration missions which require multiple instances, is to put the old instances in readonly mode by disabling the components that inject data.

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.

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.

Streams can be made ‘visible’ to the external word by two means:

- Apache ActiveMQ Artemis wrappers. There are several services that can take data from streams and publish them to Artemis addresses. Unlike the Yamcs streams which are synchronous and lightweight, Artemis addresses are asynchronous and involve more overhead. Care has to be taken with Artemis for not filling up the memory if clients are slow to read messages from the queue.

- WebSocket subscription. This can be done using the HTTP API documented separately at https://www.yamcs.org/docs/http/.

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.

Processor Clients are TM monitoring and/or TC commanding applications (Yamcs Studio, USS, MCS Tools).

Data Links

Data Links represent special components that communicate with the external world. 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.

Data Types

Yamcs supports the following high-level data types:

- A **parameter** is a data value corresponding to the observed value of a certain device. Parameters have different properties like Raw Value, Engineering Value, Monitoring status and Validity status. Currently the raw and engineering values must be of scalar types (i.e int, float, string, etc), however in the future arrays and aggregated parameters (analogous to structs in C programming language) will be supported.
- A **processed parameter** (abbreviated PP) is a particular type of parameter that is processed by an external (to Yamcs) entity. Yamcs does not contain information about how they are processed. The processed parameters have to be converted into Yamcs internal format (and therefore compatible with the Yamcs parameter types) in order to be propagated to the monitoring clients.
- A **telemetry packet** is a binary chunk of data containing a number of parameters in raw format. The packets are split into parameters according to the definitions contained in the Mission Database.
- **(Tele)commands** are used to control remote devices and are composed of a name and a list of arguments. The commands are transformed into binary packets according to the definition in the Mission Database.
- An **event** is a data type containing a source, type, level and message used by the payload to log certain kind of events. Yamcs generates internally a number of events. In order to extract events from telemetry, a special component called *Event Decoder* has to be written.

The high-level data types described above are modelled internally on a data structure called *tuple*. A tuple is a list of (name, value) pairs, where the names are simple strings and the values being of a few predefined basic data types. The exact definition of the Yamcs high-level data types in terms of tuple (e.g. a telemetry packet has the attributes gentime(timestamp), rectime(timestamp), packet(binary), etc) is currently hard-coded inside the java source code. In the future it might be externalised in configuration files to allow a certain degree of customisation.

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 three different conceptual levels:

1. **Global services** provide functionality across all instances.
2. **Instance services** provide functionality for one specific instance.
3. **Processor services** provide functionality for one specific processor.

1.3. Installation

Prerequisites

Yamcs Server runs on 64-bit Linux.

RAM	>= 1Gb
HD	>= 500Gb (dependent on amount of data archived)
Java Runtime Environment (JRE)	>= version 1.8

Install Manually

Yamcs Server software is packaged in RPM format. To install:

```
$ rpm -U yamcs-version.noarch.rpm
```

This command also works for upgrading. If a configuration file (in the etc directory) has been updated with regard to the previous installed version, the old files will be saved with the extension .rpmsave. The user then has to inspect the difference between the two versions and to implement the newly added options into the old configuration files.

To uninstall Yamcs Server use:

```
$ rpm -e yamcs
```

Note that this will also remove the yamcs user.

Install from Repository

Yamcs Server packages are distributed via yum and APT. Configure the Yamcs repository appropriate to your distribution following the repository instructions.

RPM (RHEL, Fedora, CentOS)

Install via dnf (or yum on older distributions)

```
$ dnf check-update
$ sudo dnf install yamcs
```

RPM (SLE, openSUSE)

```
$ sudo zypper refresh
$ sudo zypper install yamcs
```

File Layout

After installing the rpms, the following directories are created under /opt/yamcs:

bin	Contains shell scripts for starting the different programs
cache	Contains cached serialized java files for the Mission Database. This has to be writable by the user yamcs
etc	Contains all the configuration files
lib	Contains the jars required by Yamcs. lib/ext is where extensions reside
log	Contains the log files of Yamcs. It has to be writable by the user yamcs
mdb	Empty directory where the mission database has to be located.

In addition to the default Yamcs package, in order to get a running server, the yamcs-simulation rpm can also be installed:

```
$ rpm -U yamcs-simulation-version.noarch.rpm
```

This package will provide default configuration files and MDB for running a simple simulation of a UAV.

In addition to the directories mentioned above, yamcs also uses /storage/yamcs-data to store the data (telemetry, telecomand, event archive). This directory has to be writable by the user yamcs. The location of the data directory can be changed by editing /opt/yamcs/etc/yamcs.yaml

In addition to the default Yamcs package, there are other proprietary extensions. For example:

- yamcs-cdmcs Provides loading of TM/TC directly from CD-MCS MDB or from SCOE files (only TM) and also provides the CORBA (CIS) protocol for communicating with USS and MCS Tools
- yamcs-dass Provides TM/TC receivers/senders via the DaSS protocol
- yamcs-spell Provides the modules required to use SPELL with Yamcs. SPELL is a procedure execution environment developed by SES.

The extensions are not part of the Yamcs open-source release. If you are interested in using them, please contact Space Applications Services.

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://www.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.

Starting Yamcs Server

Normally Yamcs Server should be configured to start automatically on boot via/etc/init.d/yamcs-server. The command will automatically run itself as a lower privilege user (username yamcs), but must initially be run as root for this to happen. Yamcs Server can be started and stopped as a service via commands such as service yamcs-server start and service yamcs-server stop. These commands use the init.d script and will run Yamcs as the appropriate user. It is also possible to directly use the script /opt/yamcs/bin/yamcsd, but use of the service command is preferred.

Chapter 2. Mission Database

The Yamcs Mission Database is composed of a hierarchical structure, each node in the hierarchy being an XTCE SpaceSystem. Each SpaceSystem contains the following data:

- Telemetry
- Telecommands
- Parameters
- Algorithms

For faster access, the database is cached serialized on disk in the cache directory. The cached mission 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.

Different loaders are possible for each node in the hiearchy. A loader can load a node and its subnodes (but cannot load two parallel nodes).

```
refmdb:
  - type: "sheet"
    spec: "mdb/refmdb-ccsds.xls"
    subLoaders:
      - type: "sheet"
        spec: "mdb/refmdb-subsys1.xls"

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

2.1. Spreadsheet Loader

The spreadsheet loader loads mission database definitions from excel spreadsheet. Only excel prior to Excel 2007 are supported (.xls files not .xlsx).

Multiple Space Systems support

Since version 5.4, the spreadsheet definition supports loading from one Excel file a hierarchy composed of multiple space systems. Until version 5.3 this was only possible by defining multiple Excel files (one per subsystem) and defining the hierarchy in etc/mdb.yaml. Also until version 5.3 the loader forced some sheets to always be present (e.g. Containers). From version 5.4 only the General sheet has to be present, all the other ones are optional.

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. 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.

Conventions

- All numeric values can be entered as decimals or as hexadecimals (with prefix 0x)
- Although column names are used for reference below, columns must not be reordered

A number of mandatory named sheets are described as part of this specification, though authors may add their own sheets and still use the spreadsheet file as the reference MDB.

Rules for parameter/conainer reference lookup

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

- The reference can use UNIX like directory access expressions (../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 exact result of the lookup depends of course on the exact tree configuration in mdb.yaml”

General Sheet

This sheet must be named “General”, and the columns described must not be reordered.

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

Containers Sheet

This sheet must be named “Containers”, and the columns described must not be reordered. 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.

Parameters Sheet

This sheet must be named ending with “Parameters”, and the columns described must not be reordered. The sheet contains parameter (sometimes called measurements) information. Any number of sheets ending with “Parameters” can be present and they all have the same structure. Each parameter has a so called “DataSource” (as per XTCE) which is not immediately configured. However by historical convention:

- DerivedParameters contains all parameter whose data source is set to "DERIVED" - these are usually results of algorithm computations.
- LocalParameters contains all parameters whose data source is set to "LOCAL" - these are parameters that can be set by the user using the Yamcs API
- All other parameter sheets contain parameters whose data source is set to "TELEMETERED" - these are parameters received from remote devices

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	See below for all supported raw types
eng type	See Engineering Types
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.

Encoding and Raw Types

Raw types describe how the parameter is encoded in the raw packet. All types are case-insensitive.

Raw Type	Encoding	Description
uint	unsigned(<n&gt, <BE|LE>)	Unsigned integer. ● n is size in bits. ● LE = little endian, BE = big endian.
	<n>	shortcut for unsigned(<n>,BE)
int	twosComplement(<n>, <BE|LE>)	two's complement encoding ● n is size in bits. ● LE = little endian, BE = big endian.
	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 is size in bits. ● LE = little endian, BE = big endian.
	<n>	shortcut for twosComplement(<n>,BE)
float	ieee754_1985(<n>, <BE|LE>)	IEE754_1985 encoding ● n is size in bits. ● LE = little endian, BE = big endian.
	<n>	shortcut for iee754_1985(<n>,BE)
boolean	<empty>	the encoding has to be empty - 1 bit is assumed.

string	fixed(<n>, <charset>)	fixed size string ● n is the size in bits of the string. Only multiple of 8 supported. ● charset is one of the charsets supported by java (UTF-8, ISO-8859-1, etc). If not specified, it is by default UTF-8 The string has to start at a byte boundary inside the container.
	PrependedSize(<x>, <charset>)	The size of string in bytes is specified in first x bits of string - the size itself will not be part of the string. ● x is the size in bits of the size tag ● charset is as defined above. Note that while x can be any number of bits<=32, the string has to start at a byte boundary.
	terminated(<0xBB>, <charset>)	terminated string - pay attention to the parameters following this one; if the terminator is not found all the buffer will be consumed; ● 0xBB specifies a byte that is the string terminator ● charset is as defined above.
binary	fixed(<n>)	fixed size byte array. ● n size of the array in bits. It has to be multiple of 8 and the parameter has to start at a byte boundary.
	PrependedSize(<x>)	byte array whose size in bytes is specified in the first x bits of the array - the size itself will not be part of the raw value. ● x is the size in bits of the size tag Note that while x can be any number of bits<=32, the byte array has to start at a byte boundary.
	<n>	shortcut for fixed(<n>)
<any of the above>	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 will use for optimizing the access to the parameters following this one. ● algorithm the name of the algorithm - it has to be defined in the algorithm sheet

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 Raw 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	62 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 corrersponds 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

Calibration Sheet

This sheet must be named “Calibration”, and the columns described must not be reordered. The 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*

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.xtceproc.jecf;
public class Expression665372494 implements org.yamcs.xtceproc.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)
```

Algorithms Sheet

This sheet must be named “Algorithms”, and the columns described must not be reordered. The 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 - note that this requires the presence of jython.jar in the Yamcs lib or lib/ext directory (it is not delivered together with Yamcs). ● 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	Reference name of a parameter. See above on how this reference is resolved. 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 will be used, which may lead to runtime errors depending on the naming conventions of the applicable script language. Note that a unique name will be required in this column, when multiple instances of the same parameter are inputted.

JavaScript algorithms

A full function body is expected. The body will be encapsulated in a javascript function like

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

The in_n and outX are to be names given in the spreadsheet column *name used in the algorithm*.

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 .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 for some algorithms (e.g. command verifiers) need to return a value (rather

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 indeted with 4 characters.

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

Java algorithms

The algorithm text is a class name with optionally parantheses enclosed string that is parsed into an object by an yaml parser. Yamcs will try to locate the given class who must be implementing the org.yamcs.algorithms.AlgorithmExecutor interface and will create an object with a constructor with three paramethers:

```
<Constructor>(Algorithm, AlgorithmExecutionContext, Object arg)
```

where arg is the argument parsed from the yaml.

If the optional argument is not present in the algorithm text definition, then the class constructor should only have two parameters. The abstract class org.yamcs.algorithms.AbstractAlgorithmExecutor 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.xtceproc.DataDecoder interface instead (see below).

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 may 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.

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

These algorithms have to implement the interface org.yamcs.xtceproc.DataDecoder.

Example Definition

algo name	language	text	trigger	in/out	param name	instance
my_avg	JavaScript	r.value = (a.value + b.value + c.value) / 3;	OnInputParameterUpdate			
				in	/MY_SS/some_temperature	-2
				in	/MY_SS/some_temperature	-1
				in	/MY_SS/some_temperature	0
				out	/MY_SS/avg_out	

Example Definition for a command verifier algorithm

algo name	language	text	trigger	in/out	param name
alg_verif_completed	JavaScript	if((receivedCmdId.value==sentCmdId.value) && (receivedSeqNum.value==sentSeqNum.value) && (stage.value==2)) { if(result.value==0) return true; else return false;}			
				in	/yamcs/cmd/arg/
				in	/yamcs/cmdHist/
				in	avc_command_s
				in	avc_command_i
				in	avc_command_e
				in	avc_command_r

Alarms Sheet

This sheet must be named “Alarms”, and the columns described must not be reordered. The 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

Example Definition

param name	context	rep	min.v	watch		warning		distress		critical		severe	
				type	val	type	val	type	val	type	val	type	val
int_para				low	-11	low	-22	low	-33				
				high	30	high	40	high	50	high	60	high	70
	other_para = 4		3	high	40	high	50			high	70		
enum_para				state	ST1	state	ST2			state	ST4		
						state	ST3						

Commands Sheet

This sheet must be named “Commands”, and the columns described must not be reordered. The 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 is 0
encoding	How to convert the raw value to binary. The supported encodings are listed in the table below.
eng type	Dngineering 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

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.

Raw Type	Encoding	Description

uint	unsigned(<n&gt, <BE|LE>)	Unsigned integer. ● n is size in bits. ● LE = little endian, BE = big endian.
	<n>	shortcut for unsigned(<n>,BE)
int	twosComplement(<n>, <BE|LE>)	two's complement encoding ● n is size in bits. ● LE = little endian, BE = big endian.
	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 is size in bits. ● LE = little endian, BE = big endian.
	<n>	shortcut for twosComplement(<n>,BE)
float	ieee754_1985(<n>, <BE|LE>)	IEE754_1985 encoding ● n is size in bits. ● LE = little endian, BE = big endian.
	<n>	shortcut for iee754_1985(<n>,BE)
boolean	<empty>	the encoding has to be empty - 1 bit is assumed.
string	fixed(<n>, <charset>)	fixed size string ● n is the size in bits of the string. Only multiple of 8 supported. ● charset is one of the charsets supported by java (UTF-8, ISO-8859-1, etc). If not specified, it is by default UTF-8
	PrependedSize(<x>, <charset><m;&gt)	The size of string in bytes is specified in first x bits of string - the size itself will not be part of the string. ● x is the size in bits of the size tag ● charset is as defined above. ● m if specified, it is the minimum size in bits of the encoded value. Note that the prepended 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.
	<n>	shortcut for fixed(<n>)

	terminated(<0xBB>, <charset><m;&gt)	terminated string; ● 0xBB specifies a byte that is the string terminator ● charset is as defined above. ● m if specified is the minimum size in bits of the encoded value. Note that the termination character 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.
binary	fixed(<n>)	fixed size byte array. ● n size of the array in bits. It has to be multiple of 8 and the argument has to start at a byte boundary.
	PrependedSize(<x>)	byte array whose size in bytes is specified in the first x bits of the array - the size itself will not be part of the raw value. ● x is the size in bits of the size tag Note that while x can be any number of bits<=32, the byte array has to start at a byte boundary.
	<n>	shortcut for fixed(<n>)

Command Options Sheet

This sheet must be named “CommandOptions”, and the columns described must not be reordered. 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 (e.g. Yamcs Studio) to raise the awarness 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.

Command Verification Sheet

The Command verification sheets 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.

Change Log Sheet

This sheet must be named “ChangeLog”, and the columns described must not be reordered. This sheet contains the list of the revision made to the MDB.

2.2. XTCE Loader

This loader reads an MDB saved in XML format compliant with the XTCE specification. For more information about XTCE, see http://www.xtce.org.

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 spec attribute.

General

Yamcs uses XTCE data structures internally as much as possible, following the XTCE v1.2. Since the version 1.2 of XTCE is not yet (as of June-2018) available, the implementation has been based on various drafts found on the Internet, notably on the xtcetools project (whose author is one of the main contributors to the standard). However not all parts of the standard are supported. This chapter presents an overview of the not supported features and details when the implementation might differ from the standard. All the features that are not mentioned in this chapter should be supported; if you encounter a problem please submit an issue.

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 compatibible 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 project mentioned above to validate your XML file.

The following concepts are not supported at all:

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

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

changeThreshold is 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 sysstems (e.g. SCOS-2k) which allows this - however it is not allowed by XTCE which enforces sizeInBitsOfSizeTag > 0.

Data Types

ValidRange

Not supported for any parameter type.

BooleanDataType

In XTCE, each BooleanDataType has a string representation. In Yamcs the value is mapped to a org.yacms.parameter.BooleanValue or the protobuf equivalnet 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 is not supported.
- SystemName is not supported.
- TimeAssociation is not supported.

Containers

BinaryEncoding not supported in the container definitions.

StringParameterType

Alarms are not supported.

Commanding

Aggregates and Arrays are not supported for commands (they are for telemetry).

ArgumentRefEntry

- IncludeCondition is not supported
- RepeatEntry is not supported

2.3. Empty Node

This “loader” allows to create an empty node (SpaceSystem) with a given name. The loader has been added in version 4.8.2 of Yamcs.

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

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

2.4. SCOS2K Loader

The SCOS2K loader loads MIB definitions as defined by SCOS-2000. To use it, you need to compile the project yamcs-scos2k and copy the jar into the yamcs lib/ext directory.

Then you can add the following in the mdb.yaml:

```
scos-mib:
  - type: "org.yamcs.scos2k.MibLoader"
    args:
        path: "/path/to/ASCII/"
        # byte offset from beginning of the packet where the type and subType are read from
        typeOffset: 7
        subTypeOffset: 8
        epoch: "UNIX"
```

Known problems and limitations

The SCOS-2K loader has been implemented as a proof of concept and only subjected to limited testing by loading a TERMA TSC test database and comparing the TM/TC decoding/encoding with the TSC. A number of known limitations are documented below.

To be fixed as soon as someone shows some interest in this project (please submit an issue if you want this fixed):

- the loader does not detect when the files have changed and does not reload the database. This is because it looks at the date of the ASCII directory. As a workaround you can either remove the serialized MDB from ~/.yamcs or /opt/yamcs/cache or run “touch ASCII” to change the date of the directory.

- command verifiers are currently not loaded

Probably not immediate priority:

- delta monitoring checks (OCP_TYPE=D) not supported
- event generation (OCP_TYPE=E) not supported
- status consistency checks (OCP_TYPE=C, PCF_USCON=Y) not supported
- arguments of type “Command Id” (CPC_CATEG=A) are not supported. These can be used to insert one command inside another one. Instead a binary command argument is being used.
- SCOS 2K allows multiple command arguments with the same name. This is not supported in Yamcs (and in XTCE) so duplicate arguments are renamed arg_ with n increasing for each argument.
- no command stack support.

2.5. CDMCS MDB Loader

TM/TC

This loader loads the telemetry/telecommand definition directly from the Oracle using the oracle jdbc driver. The relevant configuration file is etc/cdmcs-mdb.yaml.

This configuration file contains, next to the username/password used to connect to the database, the path and the version of the CCU that will be loaded and also the testConfiguration (an end item of type EGSE_TEST_CONFIGURATION).

Based on the CCU parameters and on the opsname of the testConfiguration (the test configuration can only be specified through its opsname, so the opsname must exist and be unique.), Yamcs can determine the following three attributes which are used as attributes of the XTCE header:

- CCU Internal Version - this is a number uniquely identifying the CCU and the CCU version
- Test Configuration SID - this is a number uniquely identifying the test configuration
- Consistency Date - this is the time when the configured CCU has been last modified.

Please refer to Telemetry Processing and to Commanding for details on the loading of the MDB end items and on the mapping to Yamcs structures.

The configuration parameter checkForUpdatedMdb configures Yamcs to check or not the Oracle database for modified versions of the MDB. If the MDB cannot be loaded from the serialized file, the Oracle database is checked nevertheless.

This option is useful for working offline. However if it is set to false, Yamcs will never read new versions of the database, and if the database is modified and SCOE files generated, MCS Tools will refuse to load the SCOE files (it will want old ones corresponding to the saved Yamcs database).

The packet description in CD-MCS MDB is spread over different structures. When read into Yamcs, they are converted into the XTCE structures as follows:

1. A generic sequence container named “ccsds” is created. This will be the root of the hierarchy. Three parameters are added to this sequence container:

CCSDS_APID	the APID in the CCSDS primary header
CCSDS_SH_PACKETTYPE	the packet type in the CCSDS secondary header
CCSDS_PACKETID	the packet type in the CCSDS secondary header

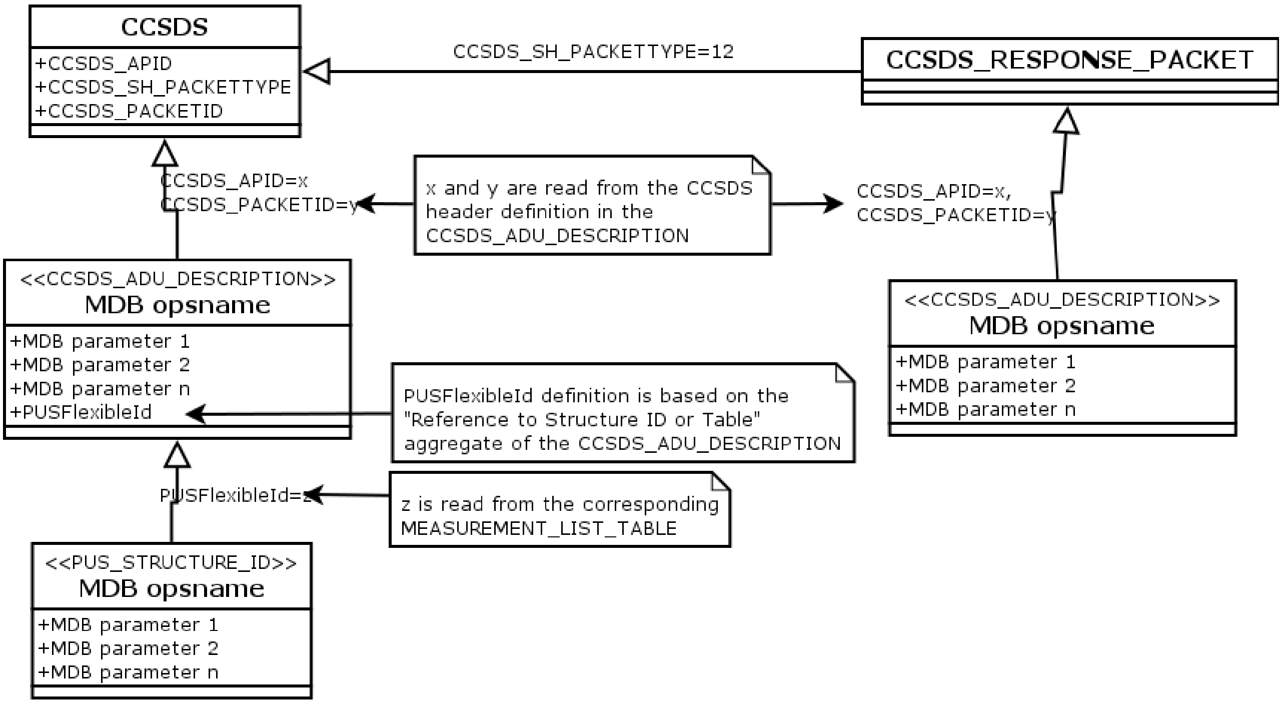
2. A generic sequence container named CCSDS_RESPONSE_PACKET inheriting from the “CCSDS” container is created. The inheritance condition is CCSDS_SH_PACKETTYPE=12(Response_packet). This packet can be used by the CIS clients which want to subscribe to all the CCSDS response packets (for example the cmd-history).

3. All the command responses (CCSDS_ADU_DESCRIPTION which have CCSDS Secondary Header set to CCSDS_RESPONSE_PACKET) are set to inherit the CCSDS_RESPONSE_PACKET container defined above. The inheritance condition is set on the

CCSDS_APID, CCSDS_PACKETID parameters.

4. All others CCSDS_ADU_DESCRIPTION are set to inherit directly the root container CCSDS. The inheritance condition is set also on the CCSDS_APID and CCSDS_PACKETID parameters.

5. For each CCSDS_ADU_DESCRIPTION that contains a “Reference To a Structure ID or Table” pointing to an end item of type MEASUREMENT_LIST_TABLE, an additional integer parameter is created containing the definition of the “Flexible ID” as defined in this aggregate. Then for each end item of type PUS_STRUCTURE_ID referred in the MEASUREMENT_LIST_TABLE, a sequence container is created in Yamcs, set to inherit the original CCSDS_ADU_DESCRIPTION with the inheritance condition on the Flexible ID as defined in the MEASUREMENT_LIST_TABLE.



PP Loader

The PP loader scans a configured CD-MCS MDB for all the end items of type UMI_MAPPING_TABLE. The first part of the Opsname (string before the underscore) is used as group name.

Chapter 3. Telemetry Processing

3.1. Packet Telemetry

The Yamcs Server implements a subset of the XTCE (XML Telemetric and Command Exchange) for telemetry processing. Only the concepts defined by the standard are supported.

For information about XTCE, please refer to http://www.xtce.org. These sections detail only the XTCE types implemented in Yamcs.

Sequence Containers

Sequence containers are the equivalent of packets in the usual terminology, or ADU in the MDB terminology.

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

Container aggregation

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

- Parameter Entries - these point to normal parameters.
- Container Entries - these point 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.

Container inheritance

Sequence containers can point to another sequence 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 which has the apid=943 and the packetid=0x1300abcd).

Little Endian Parameter Encoding

Yamcs does not currently support the XTCE way of describing byte ordering for parameter encoding.

The only alternative byte order supported is little endian. For parameters occupying entire bytes, there is no doubt on what this means. However, for parameters which occupy only part 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, assuming that on an x86 CPU we have the following structure in C:

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

Would result, when converted to network order, in 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

3.2. Algorithms

Yamcs supports the XTCE notion of *algorithms*. 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 any JSR-223 scripting language. The preferred language is specified in the instance configuration file, and applies to all algorithms within that instance. By default Yamcs ships with support for JavaScript algorithms since the standard Oracle Java distribution contains 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 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: LOW or HIGH.

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.

Triggers

Algorithms can trigger on two conditions:

- Whenever a specified parameter is updated
- Periodically (expressed in milliseconds)

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.

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, …

Being able to split the code in different user libraries is merely a user convenience. For all Yamcs cares, they could all be merged together in one big file.

Algorithm Scope

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

An algorithm's scope, however, will be shared across multiple algorithm runs. This feature 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).

Sharing State

If some kind of a shared state is required between multiple algorithms, the user libraries' shared scope could be (ab)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.

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).

3.3. Alarms

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 MDB definition, the user can define for each parameter on which conditions 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’).

Chapter 4. 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.

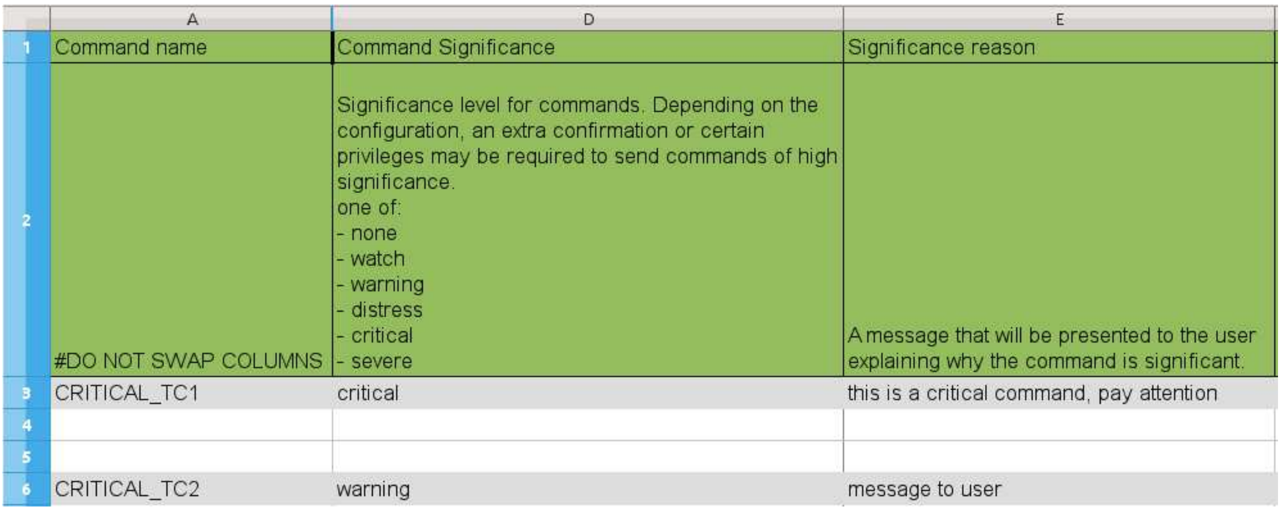
Command Significance

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

Command Queues

When a command is sent by an external user, it goes first into a queue. Privileges are checked before the command is put into the queue, so if the user doesn't have the privilege for the given command, the command will be rejected and not appear at all in the queue.

The available queues are defined in the file etc/command-queue.yaml.

```
ops:
  state: enabled
  significances: [none]

ops-critic:
  state: enabled
  significances: [watch, warning, distress, critical, severe]
  stateExpirationTimeS: 300
```

Each queue has a name, a default state and a list of roles. The commands of a user logging in with a given role will be put in the first queue for which the user has privileges. A queue can be in three different states:

Enabled	means the commands are sent immediately
Blocked	means the commands are accepted into the queue but need to be manually sent
Disabled	means the commands are rejected

There is always a command-queue called ‘default’ whose state is enabled. If a command comes from a user without privilege for any of the defined queues, the command will be put in the default queue. The default queue can be redefined in etc/command-queue.yaml in order to have a different state.

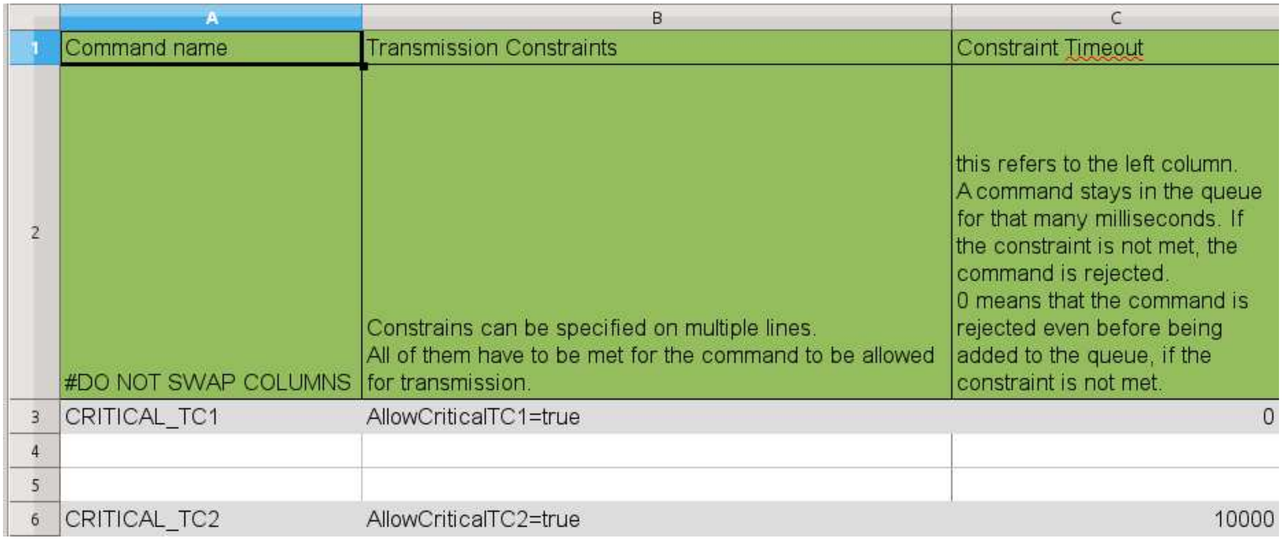
Control over the command queues, requires the ControlCommandQueue privilege.

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).

Chapter 5. Archive

The Yamcs archive is divided in two parts:

- Stream Archive stores time ordered tuples (t, v_1, v_2…v_n) where t is the time and v_1, v_2, v_n are values of various types. This is used for storing raw telemetry packets, commands, events, alarms and processed parameters. The stream archive can be seen as a row-oriented archive and is optimized for accessing entire records (e.g. a packet or a group of processed parameters).

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

Data is stored in the Stream Archive as soon as it is being received, whereas the Parmeter Archive involves some data transformation and it is filled in batches. However, for an external user, Yamcs should make the filling process invisible so data from both archives can be retrieved (almost) as soon as it been received by Yamcs.

5.1. Stream Archive

Yamcs uses streams to transfer tuples of data. A tuple has a variable number of columns, each of predefined type. The Yamcs stream archive is composed of tables that store data passing through streams.

Like streams, the tables have a variable number of columns of predefined types. In addition to that, the tables have also a primary key composed of one or more columns. The primary key columns are mandatory, a tuple that doesn’t have them will not be stored in the table.

The primary key is used to sort the data in the table. 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.

On the basic stream archive structure, Yamcs pre-defines a few table types for storing data at higher level of abstractions. These are Packet telemetry, Events, Command history, Alarms and Parameters and are described in the next sections.

Packet telemetry

The below definition (created inside the XtceTmRecorder service) will create a table that uses the generation time and sequence number as primary key. That means that if a packet has the same time and sequence number as another packet already in the archive, it will not be stored (considered duplicate).

```
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;
```

Following is a short description of the columns used:

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

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 will be partition in different RocksDB databases and column families based on the time and container name. Currently the time partitioning schema used is “YYYY/MM” which means there will be one RocksDB database per year,month. Inside that database there will be one column family for each container that is used for partitioning. Parittioning the data based on time, ensures that old data will be “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 all packets will be stored in the same partition.

Events

The below definition (created by the EventRecorder service) will create a table that 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;
```

Following is a short description of the columns used:

- gentime - is the generation time of the command set by the originator

- source - is a string representing the source of the events
- seqNum - is a sequence number provided by the event source. Each source is supposed to keep an independent sequence count for the events it generates.

Command history

The below definition (created by the CommandHistoryRecorder service) will create a table that 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;
```

Following is a short description of the columns used:

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

Alarms

The below definition (created by the AlarmRecorder service) will create a table that 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;
```

Following is a short description of the columns used:

- triggerTime - is 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 though it goes back in limits in the meanwhile)
- parameter - the fully qualified XTCE 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.

Parameters

The below definition (created by the ParameterRecorder service) will create a table that 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;
```

Following is a short description of the columns used:

- gentime - is the generation time of the command set by the originator.
- ppgroup - is a string used to group parameters. The parameters sharing the same group and the same timestamp are stored together.
- seqNum - is a sequence number supposed to be increasing independently for each group.
- rectime - is 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 that because partitioning by ppgroup is specified, this is also implicitly part of the primary key (but not stored as such in the RocksDB key).

5.2. Parameter Archive

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

- t_i - is the “generation” timestamp of the value. The “reception” timestamp is not stored in the Parameter Archive.
- ev_i - is the engineering value of the parameter at the given time.
- rv_i - is the engineering value of the parameter at the given time.
- ps_i - is the parameter status of the parameter at the given time.

The parameter status includes things like out of limits (alarms), processing status, etc. XTCE provides a mechanism through which a parameter can change its alarm ranges depending on the context. For this reason we store in the parameter status also 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. More detail about the parameter archive organization can be found in the Parameter Archive Internals. Having all the data inside one segment of the same type offers possibility for good compression especially if the values don’t change much or at all (as it is often the case).

While this structure is good for fast retrieval, it doesn’t allow updating data very efficiently and in any case not in realtime (like the stream archive does). This is why the parameter archive is filled in batch mode - data is accumulated in memory and flushed to disk periodically. The sections below explain the different filling strategies implemented.

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 stream archive and write the generated parameters to the 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’s 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 tha 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).

5.3. Parameter Archive Internals

Why not store the values on change only?

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

We can notice that 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 parameter archives only store 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 will be 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.

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 miliseconds (=~ 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 (Xtce db).

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

Column Families for storing metadata we have 2 CFs:
meta_p2pid: contains the mapping between parameter fully qualified 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 is basetimestamp (i.e. the start timetamp 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 rockdsdb files in sequence of $\text{engvalue}_{\text{segment\_1}}$,$\text{rawvalue}_{\text{segment\_1}}$,$\text{parameterstatus}_{\text{segment\_1}}$, $\text{engvalue}_{\text{segment\_2}}$. $\text{rawvalue}_{\text{segment\_2}}$...

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.

Description and implementation of the FastPFOR algorithm can be found here.

IntSegment - stores int32 or uint32 encoded same way as the time segment.

FloatSegment - stores 32 bits floating point numbers encoded using the algorithm (very slightly modified to work on 32 bits) described in the Facebook Gorilla paper

ParameterStatusSegment, **StringSegment** and **BinarySegment** are all stored either raw, as an enumeration, or run-length encoded, depeending on which results in smaller compressed size.

DoubleSegment and **LongSegment** are only stored as raw for the moment - compression is still 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).

Future work

Segment Compression Compression for DoubleSegment and the LongSegment. DoubleSegment is straightforward, for the 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. This may be implemented in a future version. Another useful feature would be to trigger the back filling automatically when gaps are filled in the stream archive. This will hopefully also be implemented in a future version (contributions welcome!).

Chapter 6. Global Services

6.1. Artemis Server

Initializes and starts an embedded instance of the Artemis messaging server. This can be used to connect streams across Yamcs installations.

Class Name

org.yamcs.artemis.ArtemisServer

Configuration

This is a global service defined in etc/yamcs.yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.artemis.ArtemisServer
```

Configuration Options

Name	Type	Description
configFile	string	Filename of the XML configuration file that contains further configuration options. Do not use an absolute path. The file must exist in the /opt/yamcs/etc folder. Default: artemis.xml.
securityManager	string	Class name of a org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager implementation. The implementation should have a no-arg constructor. If unspecified, security is not enabled.

6.2. HTTP Server

Embedded HTTP server that provides these functionalities:

- Serve HTTP API (REST and WebSocket)
- Serve the Yamcs web interface

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.

Class Name

org.yamcs.web.HttpServer

Configuration

This is a global service defined in etc/yamcs.yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.web.HttpServer
    args:
      webRoot: lib/yamcs-web
      port: 8090
      webSocket:
        writeBufferWaterMark:
          low: 32768
          high: 65536
        connectionCloseNumDroppedMsg: 5
      cors:
        allowOrigin: "*"
        allowCredentials: false
      website:
        displayScope: GLOBAL
```

Configuration Options

Name	Type	Description
port	integer	The port at which Yamcs web services may be reached. Default: 8090
webRoot	string or string[]	List of file paths that are statically served. This usually points to the web files for the built-in Yamcs web interface (lib/yamcs-web).
zeroCopyEnabled	boolean	Indicates whether zero-copy can be used to optimize non-SSL static file serving. Default: true
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.
website	map	Configure properties of the Yamcs website.

WebSocket sub-configuration

Name	Type	Description
maxFrameLength	integer	Maximum frame length in bytes. Default: 65535
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}
connectionCloseNumDroppedMsg	integer	Allowed number of message drops before closing the connection. Default: 5

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.

Name	Type	Description
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

Website sub-configuration

Name	Type	Description
displayScope	string	Where to locate displays and layouts. One of INSTANCE or GLOBAL. Setting this to GLOBAL means that displays are shared between all instances. Setting this to INSTANCE, means that each instance uses its own displays. Default: GLOBAL

6.3. Process Runner

Runs an external process. If this process goes down, a new process instance is started.

Class Name

org.yamcs.server.ProcessRunner

Configuration

This is a global service defined in etc/yamcs.yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.server.ProcessRunner
    args:
      command: "bin/simulator.sh"
```

Configuration Options

Name	Type	Description
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 cwd 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] '.

6.4. 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.

Class Name

org.yamcs.tse.TseCommander

Configuration

This is a global service defined in etc/yamcs.yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.tse.TseCommander
```

Configuration Options

Name	Type	Description
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:

Name	Type	Description
responseTermination	string	The character(s) by which the instrument delimits distinct responses. Typicall \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

In addition each driver supports driver-specific arguments:

TCP/IP

Name	Type	Description
host	string	**Required.** The host of the instrument.
port	integer	**Required.** The TCP port to connect to.

Serial Port

Name	Type	Description
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
parity	string	The parity error-detection scheme. One of odd or even. By default parity is not set.

Mission Database

The definition of TSE string commands is done in XTCE space systems resorting under /TSE. The /TSE node is added by defining org.yamcs.xtce.TseLoader 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.
  - 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	Parent	Assignments	Arguments
get_identification	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:

Command name	Parent	Assignments	Arguments
get_battery_voltage	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).

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).

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.

Chapter 7. Instance Services

7.1. Alarm Recorder

Records alarms. This service stores the data coming from one or more streams into a table alarms.

Class Name

org.yamcs.archive.AlarmRecorder

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
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.

7.2. Artemis Command History Publisher

Publish cmdhist stream data to an Artemis broker.

Class Name

org.yamcs.artemis.ArtemisCmdHistoryService

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.artemis.ArtemisCmdHistoryService
    args: [cmdhist_realtime, cmdhist_dump]
```

args must be a an array of strings indicating which streams to publish. For each stream the target address is composed as instance.stream . In the example tuples from the streams cmdhist_realtime and cmdhist_dump are published to the addresses simulator.cmdhist_realtime and simulator.cmdhist_dump respectively.

By default, messages are published to an embedded broker (in-VM). This assumes that Artemis Server was configured as a global service. In order to use an external broker you can configure the property artemisUrl in either etc/yamcs.(instance).yaml or etc/yamcs.yaml:

```
artemisUrl: tcp://remote-host1:5445
```

```
artemisUrl: tcp://remote-host1:5445
```

If defined, the instance-specific configuration is selected over the global configuration. The URL format follows Artemis conventions and is not further detailed in this manual.

7.3. Artemis Event Publisher

Publish event stream data to an Artemis broker.

Class Name

org.yamcs.artemis.ArtemisEventService

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.artemis.ArtemisEventService
    args: [events_realtime, events_dump]
```

args must be a an array of strings indicating which streams to publish. For each stream the target address is composed as instance.stream . In the example tuples from the streams events_realtime and events_dump are published to the addresses simulator.events_realtime and simulator.events_dump respectively.

By default, messages are published to an embedded broker (in-VM). This assumes that Artemis Server was configured as a global service. In order to use an external broker you can configure the property artemisUrl in either etc/yamcs.(instance).yaml or etc/yamcs.yaml:

```
artemisUrl: tcp://remote-host1:5445
```

```
artemisUrl: tcp://remote-host1:5445
```

If defined, the instance-specific configuration is selected over the global configuration. The URL format follows Artemis conventions and is not further detailed in this manual.

7.4. Artemis Parameter Publisher

Publish param stream data to an Artemis broker.

Class Name

org.yamcs.artemis.ArtemisPpService

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.artemis.ArtemisPpService
    args: [pp_realtime, pp_dump, sys_param]
```

args must be a an array of strings indicating which streams to publish. For each stream the target address is composed as instance.stream . In the example tuples from the streams pp_realtime, pp_dump and pp_dump are published to the addresses simulator.pp_realtime, simulator.pp_dump and simulator.pp_sys_param respectively.

By default, messages are published to an embedded broker (in-VM). This assumes that Artemis Server was configured as a global service. In order to use an external broker you can configure the property artemisUrl in either etc/yamcs.(instance).yaml or etc/yamcs.yaml:

```
artemisUrl: tcp://remote-host1:5445
```

```
artemisUrl: tcp://remote-host1:5445
```

If defined, the instance-specific configuration is selected over the global configuration. The URL format follows Artemis conventions and is not further detailed in this manual.

7.5. Artemis TM Publisher

Publish tm stream data to an Artemis broker.

Class Name

org.yamcs.artemis.ArtemisTmService

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.artemis.ArtemisTmService
    args: [tm_realtime, tm_dump]
```

args must be a an array of strings indicating which streams to publish. For each stream the target address is composed as instance.stream . In the example tuples from the streams tm_realtime and tm_dump are published to the addresses simulator.tm_realtime and simulator.tm_dump respectively.

By default, messages are published to an embedded broker (in-VM). This assumes that Artemis Server was configured as a global service. In order to use an external broker you can configure the property artemisUrl in either etc/yamcs.(instance).yaml or etc/yamcs.yaml:

```
artemisUrl: tcp://remote-host1:5445
```

```
artemisUrl: tcp://remote-host1:5445
```

If defined, the instance-specific configuration is selected over the global configuration. The URL format follows Artemis conventions and is not further detailed in this manual.

7.6. Command History Recorder

Records command history entries. This service stores the data coming from one or more streams into a table cmdhist.

Class Name

org.yamcs.archive.CommandHistoryRecorder

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
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.

7.7. Data Link Initialiser

Manages the various data links and creates needed streams during Yamcs start-up.

Data Links represent data connections to external sources. These connections may represent output flows (TC), input flows (TM, PP) or a combination of these. Data links that read TM and PP data receive telemetry packets or parameters and inject them into the realtime or dump TM or PP streams. Data links that send TC subscribe to a TC stream and send data to external systems.

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.

Class Name

org.yamcs.tctm.DataLinkInitialiser

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.tctm.DataLinkInitialiser

dataLinks:
  - name: tm_realtime
    class: org.yamcs.tctm.TcpTmDataLink
    args:
      stream: tm_realtime
      host: localhost
      port: 10015
  - name: tm_dump
    class: org.yamcs.tctm.TcpTmDataLink
    args:
      stream: tm_dump
      host: localhost
      port: 10115
  - name: udp_realtime
    class: org.yamcs.tctm.UdpTmDataLink
    args:
      stream: tm_realtime
      port: 5900
      maxLength: 2048
  - name: tc_realtime
    class: org.yamcs.tctm.TcpTcDataLink
    args:
      stream: tc_realtime
      host: localhost
      port: 10025
```

Each link is defined in terms of an identifying name and a class. There is also a property enabledAtStartup which allows to enable (default) or disable the TM provider for connecting to the external data source at the server start-up.

Specific data links may support additional arbitrary configuration options via the args key.

7.7.1. Artemis Parameter Data Link

Reads param data from an Artemis queue and publishes it to the configured stream.

Class Name

org.yamcs.tctm.ArtemisParameterDataLink

Configuration Options

Name	Type	Description
stream	string	Required. The stream where data is emitted
address	string	Artemis address to bind to.

7.7.2. Artemis TM Data Link

Reads tm data from an Artemis queue and publishes it to the configured stream.

Class Name

org.yamcs.tctm.ArtemisTmDataLink

Configuration Options

Name	Type	Description
stream	string	Required. The stream where data is emitted
address	string	Artemis address to bind to.
preserveIncomingReceptionTime	boolean	When true incoming reception times are preserved. When false each packet is tagged with a fresh reception timestamp. Default: false

7.7.3. 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.

Class Name

org.yamcs.tctm.FilePollingTmDataLink

Configuration Options

Name	Type	Description
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 implementation. Default is org.yamcs.tctm.IssPacketPreprocessor which applies ISS conventions.
packetPreprocessorArgs	map	Optional args of arbitrary complexity to pass to the PacketPreprocessor. Each PacketPreprocessor may support different options.

7.7.4. TCP TC Data Link

Sends telecommands via TCP.

Class Name

org.yamcs.tctm.TcpTcDataLink

Configuration Options

Name	Type	Description
stream	string	Required. The stream where command instructions are received
host	string	**Required.** The host of the TC provider
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 implementation. Default is org.yamcs.tctm.IssCommandPostProcessor which applies ISS conventions.
commandPostprocessorArgs	map	Optional args of arbitrary complexity to pass to the CommandPostprocessor. Each CommandPostprocessor may support different options.

7.7.5. 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.

Class Name

org.yamcs.tctm.TcpTmDataLink

Configuration Options

Name	Type	Description
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 implementation. Default is org.yamcs.tctm.CcsdsPacketInputStream 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 implementation. Default is org.yamcs.tctm.IssPacketPreprocessor which applies ISS conventions.
packetPreprocessorArgs	map	Optional args of arbitrary complexity to pass to the PacketPreprocessor. Each PacketPreprocessor may support different options.

7.7.6. TSE Data Link

Sends telecommands to a configured TSE Commander and reads back output as processed parameters.

Class Name

org.yamcs.tctm.TseDataLink

Configuration Options

Name	Type	Description
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

7.7.7. UDP Parameter Data Link

Listens on a UDP port for datagrams containing Protobuf encoded messages. One datagram is equivalent to a message of type org.yamcs.protobuf.Pvalue.ParameterData

Class Name

org.yamcs.tctm.UdpParameterDataLink

Configuration Options

Name	Type	Description
stream	string	Required. The stream where data is emitted
port	integer	Required. The UDP port to listen on

7.7.8. UDP TM Data Link

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

Class Name

org.yamcs.tctm.UdpTmDataLink

Configuration Options

Name	Type	Description
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
packetPreprocessorClassName	string	Class name of a PacketPreprocessor implementation. Default is org.yamcs.tctm.IssPacketPreprocessor which applies ISS conventions.
packetPreprocessorArgs	map	Optional args of arbitrary complexity to pass to the PacketPreprocessor. Each PacketPreprocessor may support different options.

7.8. Event Recorder

Records events. This service stores the data coming from one or more streams into a table events.

Class Name

org.yamcs.archive.EventRecorder

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
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.

7.9. Index Server

Supports retrieval of archive indexes and tags.

Class Name

org.yamcs.archive.IndexServer

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.archive.IndexServer
```

Configuration Options

Name	Type	Description
tmIndexer	string	Class name of a TmIndex implementation. Default is org.yamcs.archive.CcsdsTmIndex which applies CCSDS conventions.

7.10. 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.

Class Name

org.yamcs.parameterarchive.ParameterArchive

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.parameterarchive.ParameterArchive
    args:
      realtimeFiller:
        enabled: true
        flushInterval: 300 #seconds
      backFiller:
        #warmupTime: 60 seconds default warmupTime
        enabled: true
        schedule: [{startSegment: 10, numSegments: 3}]
```

This configuration enables the realtime filler flushing the data to the archive each 5 minutes, and in addition the backFiller fills the archive 10 segments (approx 700 minutes) in the past, 3 segments at a time.

```
services:
  - class: org.yamcs.parameterarchive.ParameterArchive
    args:
      realtimeFiller:
        enabled: false
      backFiller:
        enabled: true
        warmupTime: 120
        schedule:
          - {startSegment: 10, numSegments: 3}
          - {startSegment: 2, numSegments: 2, interval: 600}
```

This configuration does not use the realtime filler, but instead performs regular (each 600 seconds) back-fillings of the last two segments. 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.

7.11. 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.

Parameters extracted from packets are usually not stored in pp. Instead Yamcs provides a different service called the Parameter Archive which is specially optimized for data retrieval.

Class Name

org.yamcs.archive.ParameterRecorder

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
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.

Configuration Options

Name	Type	Description
streams	list of strings	The streams to record. When unspecified, all param streams defined in streamConfig are recorded.

7.12. Processor Creator Service

Creates persistent processors owned by the system user.

Class Name

org.yamcs.ProcessorCreatorService

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.ProcessorCreatorService
    args:
      name: realtime
      type: realtime
```

Configuration Options

Name	Type	Description
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

7.13. 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.

Class Name

org.yamcs.archive.ReplayServer

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.archive.ReplayServer
```

7.14. System Parameters Collector

Collects system parameters from any Yamcs component at a frequency of 1Hz. Parameter values are emitted to the sys_var stream.

Class Name

org.yamcs.parameter.SystemParametersCollector

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
services:
  - class: org.yamcs.parameter.SystemParametersCollector
    args:
      provideJvmVariables: true
```

Configuration Options

Name	Type	Description
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

7.15. XTCE TM Recorder

Records XTCE TM sequence containers. This service stores the data coming from one or more streams into a table tm.

Class Name

org.yamcs.archive.XtceTmRecorder

Configuration

This service is defined in etc/yamcs.(instance).yaml. Example from a typical deployment:

```
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.

Configuration Options

Name	Type	Description
streams	list of strings	The streams to record. When unspecified, all tm streams defined in streamConfig are recorded.

Chapter 8. Processor Services

8.1. 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.

Class Name

org.yamcs.alarms.AlarmReporter

Configuration

This service is defined in etc/processor.yaml. Example from a typical deployment:

```
realtime:
  services:
    - class: org.yamcs.alarms.AlarmReporter
```

Configuration Options

Name	Type	Description
source	string	The source name of the generated events. Default: AlarmChecker

8.2. Algorithm Manager

Executes algorithms and provides output parameters.

Class Name

org.yamcs.algorithms.AlgorithmManager

Configuration

This service is defined in etc/processor.yaml. Example:

```
realtime:
  services:
    - class: org.yamcs.algorithms.AlgorithmManager
      args:
        libraries:
          JavaScript:
            - "mdb/mylib.js"
```

Configuration Options

Name	Type	Description
libraries	map	Libraries to be included in algorithms. The map points from the scripting language to a list of file paths.

8.3. Local Parameter Manager

Manages and provides local parameters.

Class Name

org.yamcs.parameter.LocalParameterManager

Configuration

This service is defined in etc/processor.yaml. Example:

```
realtime:
  services:
    - class: org.yamcs.parameter.LocalParameterManager
```

8.4. Replay Service

Provides telemetry packets and processed parameters from the archive.

Class Name

org.yamcs.tctm.ReplayService

Configuration

This service is defined in etc/processor.yaml. Example:

```
Archive:
  services:
    - class: org.yamcs.tctm.ReplayService
```

Configuration Options

Name	Type	Description
excludeParameterGroups	list of strings	Parameter groups to exclude from being replayed.

8.5. Stream Parameter Provider

Provides parameters received from the configured param stream.

Class Name

org.yamcs.tctm.StreamParameterProvider

Configuration

This service is defined in etc/processor.yaml. Example:

```
realtime:
  services:
    - class: org.yamcs.tctm.StreamParameterProvider
      args:
        stream: "pp_realtime"
```

Configuration Options

Name	Type	Description
streams	list of strings	**Required.** The streams to read.

8.6. Stream TC Command Releaser

Sends commands to the configured tc stream.

Class Name

org.yamcs.StreamTcCommandReleaser

Configuration

This service is defined in etc/processor.yaml. Example:

```
realtime:
  services:
    - class: org.yamcs.StreamTcCommandReleaser
      args:
        stream: "tc_realtime"
```

Configuration Options

Name	Type	Description
stream	string	**Required.** The stream to send commands to.

8.7. 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 .

Class Name

org.yamcs.StreamTmPacketProvider

Configuration

This service is defined in etc/processor.yaml. Example:

```
realtime:
  services:
    - class: org.yamcs.StreamTmPacketProvider
      args:
        streams: ["tm_realtime", "tm_dump"]
```

Configuration Options

Name	Type	Description
streams	list of strings	**Required.** The streams to read.

8.8. System Parameter Provider

Provides parameters received from the sys_param stream.

Class Name

org.yamcs.parameter.SystemParametersProvider

Configuration

This service is defined in etc/processor.yaml. Example:

```
realtime:
  services:
    - class: org.yamcs.parameter.SystemParametersProvider
```

Chapter 9. Logging

Yamcs Server writes log output to the directory `/opt/yamcs/log`.

yamcs-server.log.x

Log files usually provide the best debug information. These contain log entries that are emitted by any of the Yamcs components with a fine level of detail.

Log files are rotated at 20MB with a maximum of 50 files. The theoretic maximum of disk space is therefore 1GB. `x` is a sequence number. The lower the number, the more recent the logs. The most recent log file can always 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.

yamcs-server.out.x

Out files are directly captured from the process standard output and error streams. The logging level is typically less detailed then with `yamcs-server.log.x`, but the files may contain stdout and stderr output which does not make use of Yamcs' logging system.

Out files are rotated over a maximum of 5 files. There is no size restriction on the file, but since the logging is not so detailed, the files do not grow very large. `x` is a sequence number. The lower the number, the more recent the logs. The most recent out file can always be found at `/opt/yamcs/log/yamcs-server.out.0`. Note that when Yamcs Server is restarted the out files will always rotate.

Configuration

The logging properties of Yamcs Server may be adjusted to your specific situation. This is done by modifying the file `/opt/yamcs/etc/logging.yamcs-server.properties`. The file structure is defined by the standard Java logging framework and requires a bit of background with Java development. There are two handlers. A FileHandler defines the properties used for logging to `/opt/yamcs/log/yamcs-server.log.x`. A ConsoleHandler can be used to tweak output for logging to `/opt/yamcs/log/yamcs-server.out.x`. The rotation of out files is not configured in this file, since this occurs at the level of the init system where the Yamcs process is managed.

Yamcs comes with different log formatters that can be useful in different situations:

- org.yamcs.CompactFormatter outputs one line per log entry and contains detailed datetime information, thread id, severity, logger name (typically the class of the originating component), a log message and finally an optional stack trace.

- org.yamcs.ConsoleFormatter outputs one line per log entry and is actually more compact than `CompactFormatter`. It is especially suited for development of Yamcs or its extensions. Each log entry contains short time information, thread id, short class name, a log message and optionally a stack trace. Some entry fields make use of ANSI color codes for colorized display inside the developer's terminal.

Chapter 10. 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.

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
ModifyCommandHistory	Allows to modify command history
ControlCommandQueue	Allows to manage command queues
Command	Allows to issue any command
GetMissionDatabase	Allows to read the entire MDB
ControlArchiving	Allows to manage archiving properties of Yamcs
ControlLinks	Allows to control the lifecycle of any data 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
WriteEvents	Allows to manually post events
WriteTables	Allows to manually add records to tables
ReadTables	Allows to read tables

Yamcs extensions may support additional system privileges.

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 particular command
CommandHistory	Allows access to the command history of a particular command
InsertCommandQueue	Allows to insert commands to a particular queue
ManageBucket	Allow control over 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 particular parameter

Yamcs extensions may support additional object privileges.

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.

AuthModules

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:

YAML AuthModule

Reads Yaml files to verify the credentials of the user, or assign privileges.

LDAP AuthModule

Attempts to bind to LDAP with the provided credentials. Also capable of reading privileges for the user.

SPNEGO AuthModule

Supports authenticating against a Kerberos server. This module includes extra support for Single-Sign-On via Yamcs web interface.

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.

A special note on *roles*. Yamcs itself does not require roles nor does it keep track of roles on the User object. Permissions are always verified via user privileges. Specific AuthModules may however introduce roles as a convenience to group sets of privileges together.

Configuration

Example from a typical deployment:

```
enabled: true

# Implicit user when security is _not_ enabled
unauthenticatedUser:
  username: admin
  superuser: true

authModules:
  - class: org.yamcs.security.YamlAuthModule
    config:
      hasher: org.yamcs.security.PBKDF2PasswordHasher
```

These options are supported:

enabled

Whether security is enabled. If false then Yamcs will not require users to login and will assume that everybody shares a single account defined under the unauthenticatedUser property.

unauthenticatedUser

Configures the user details of the default user. This property is only considered when enabled is set to false

authModules

List of AuthModules that particpate in the login process. Each AuthModule may support custom configuration options which can be defined under the config key.

10.1. YAML AuthModule

The YAML AuthModule supports authentication and authorization of users via YAML files available directly in the etc/ folder.

Class Name

org.yamcs.security.YamlAuthModule

Configuration Options

Name	Type	Description
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
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.

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.

```
admin:
  password: somepassword
  superuser: true

someuser:
  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:

```
yamcs 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.

roles.yaml

This file defines which privileges belong to which roles.

```
Operator:
  ReadParameter: [".*"]
  WriteParameter: []
  ReadPacket: [".*"]
  Command: [".*"]
  InsertCommandQueue: ["ops"]
  System:
    - ControlProcessor
    - ModifyCommandHistory
    - ControlCommandQueue
    - Command
    - GetMissionDatabase
    - 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.

All keys are optional so the simplest role definition is simply:

```
EmptyRole:
```

10.2. LDAP AuthModule

The LDAP AuthModule supports authentication and authorization of users via the LDAP protocol.

This module adds privileges to users through the use of roles: a user has specific roles, and some role is required for a specific privilege. All the user, role and privilege definitions are looked up in the LDAP database. Yamcs reads only LDAP objects of type groupOfNames.

The algorithm used to build the list of user privileges is as follows:

- From the path configured by rolePath find all the roles associated to the user. The roles defined in LDAP must contain references using the member attribute to objects member=uid=username from the userPath.
- For each role found previously, do a search in the corresponding system, tc, tm packet or tm parameter path using the match member=cn=role_name. The cn of the matching entries is used to build the list of privileges that the user has.

This class can be stacked with other AuthModules such that it is responsible for either authentication or authorization. In the case of authorization read-only access to the LDAP database is assumed.

Class Name

org.yamcs.security.LdapAuthModule

Configuration Options

Name	Type	Description
host	string	Required. The LDAP host
userPath	string	Required. The path to user definitions
rolePath	string	The path to role definitions
systemPath	string	The path to system privileges
tmParameterPath	string	The path to ReadParameter object privileges
tmPacketPath	string	The path to ReadPacket object privileges
tcPath	string	The path to Command object privileges

10.3. SPNEGO AuthModule

The SPNEGO AuthModule supports authentication of users via an external Kerberos server. It does not support authorization and must therefore be stacked together with another AuthModule.

Class Name

org.yamcs.security.SpnegoAuthModule

Configuration Options

Name	Type	Description
krbRealm	string	Accept only users from this realm
stripRealm	boolean	Whether to strip the realm from the username (e.g. 'user@REALM' becomes just 'user'). Use this only when krbRealm is also set. Default: false
krb5.conf	string	Absolute path to the applicable krb5.conf file.
jaas.conf	string	Absolute path to the applicable jaas.conf file.

The jaas.conf file must contain login modules called UserAuth and Yamcs. Details are beyond the scope of this manual.

Chapter 11. Programs

11.1. yamcsadmin

NAME

yamcsadmin – Tool for local Yamcs administration

SYNOPSIS

yamcsadmin [--etc-dir DIR] COMMAND

OPTIONS

--etc-dir DIR

Override default Yamcs configuration directory

-h, --help

Show usage

-v, --version

Print version information and quit

COMMANDS

backup
Perform and restore backups

confcheck
Check the configuration files of Yamcs

parchive
Parameter Archive operations

password-hash
Generate password hash for use in users.yaml

rocksdb
Provides low-level RocksDB data operations

xtcedb
Provides information about the XTCE database

11.1.1. yamcsadmin backup

NAME

yamcsadmin backup – Perform and restore backups

SYNOPSIS

yamcsadmin backup COMMAND

COMMANDS

create
Create a new backup

delete
Delete a backup

list
List the existing backups

restore
Restore a backup.

### yamcsadmin backup create

NAME

yamcsadmin backup create – Create a new backup

SYNOPSIS

yamcsadmin backup create [--dbDir DIR] [--backupDir DIR]

OPTIONS

--dbDir DIR

Database directory.

--backupDir DIR

Backup Directory.

### yamcsamdin backup delete

NAME

yamcsadmin backup delete – Delete a backup

SYNOPSIS

yamcsadmin backup delete [--backupDir DIR] [--backupId ID]

OPTIONS

--backupDir DIR

Backup Directory.

--backupId ID

Backup ID.

### yamcsadmin backup list

NAME

yamcsadmin backup list – List the existing backups

SYNOPSIS

yamcsadmin backup list [--backupDir DIR]

OPTIONS

--backupDir DIR

Backup Directory.

### yamcsadmin backup restore

NAME

yamcsadmin backup restore – Restore a backup

SYNOPSIS

yamcsadmin backup restore [--backupDir DIR] [--backupId ID] [--restoreDir DIR]

DECRIPTION

Note that backups can only be restored when Yamcs is not running.

OPTIONS

--backupDir DIR

Backup Directory.

--backupId ID

Backup ID. If not specified, defaults to the last backup

--restoreDir DIR

Directory where to restore the backup

11.1.2. yamcsadmin confcheck

NAME

yamcsadmin confcheck – Check the configuration files of Yamcs

SYNOPSIS

yamcsadmin confcheck [--no-etc] [DIR]

POSITIONAL ARGUMENTS

[DIR]

Use this directory in preference for loading configuration files. If --no-etc is specified, all configuration files will be loaded from this directory. Note that the data directory (yamcs.yaml dataDir) will be changed before starting the services, otherwise there will be RocksDB LOCK errors if a yamcs server is running.

OPTIONS

--no-etc

Do not use any file from the default Yamcs etc directory. If this is specified, the argument config-dir becomes mandatory.

11.1.3. yamcsadmin parchive

NAME

yamcsadmin parchive – Parameter Archive operations

SYNOPSIS

yamcsadmin parchive --instance INSTANCE COMMAND

OPTIONS

--instance INSTANCE

Yamcs instance.

COMMANDS

print-pid

Print parameter name to parameter id mapping

print-pgid

Print parameter group compositions

### yamcsadmin parchive print-pid

NAME

yamcsadmin parchive print-pid – Print parameter name to parameter id mapping

SYNOPSIS

yamcsadmin parchive print-pid

### yamcsadmin parchive print-pgid

NAME

yamcsadmin parchive print-pgid – Print parameter group compositions

SYNOPSIS

yamcsadmin parchive print-pgid

11.1.4. yamcsadmin password-hash

NAME

yamcsadmin password-hash – Generate password hash for use in users.yaml

SYNOPSIS

yamcsadmin password-hash

DESCRIPTION

Promps 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.

11.1.5. yamcsadmin rocksdb

NAME

yamcsadmin rocksdb – Provides low-level RocksDB data operations

SYNOPSIS

yamcsadmin rocksdb COMMAND

COMMANDS

compact

Compact rocksdb database

bench

Benchmark rocksdb storage engine

### 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).

### 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 wll 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.

11.1.6. yamcsadmin xtcedb

NAME

yamcsadmin xtcedb – Provides information about the XTCE database

SYNOPSIS

yamcsadmin xtcedb [-f FILE] COMMAND

OPTIONS

-f FILE

Use this file instead of default mdb.yaml

COMMANDS

listConfigs

List the MDB configurations defined in mdb.yaml

print

Print the contents of the XTCE DB

verify

Verify that the XTCE DB can be loaded

### yamcsadmin xtcedb listConfigs

NAME

yamcsadmin xtcedb listConfigs – List the MDB configurations defined in mdb.yaml

SYNOPSIS

yamcsadmin xtcedb listConfigs

### yamcsadmin xtcedb print

NAME

yamcsadmin xtcedb print – Print the contents of the XTCE DB

SYNOPSIS

yamcsadmin xtcedb print CONFIG

POSITIONAL ARGUMENTS

CONFIG

Config name.

### yamcsadmin xtcedb verify

NAME

yamcsadmin xtcedb verify – Verify that the XTCE DB can be loaded

SYNOPSIS

yamcsadmin xtcedb verify CONFIG

POSITIONAL ARGUMENTS

CONFIG

Config name.

11.2. yamcsd

NAME

yamcsd – Yamcs Server

SYNOPSIS

yamcsd

DESCRIPTION

yamcsd is the main program in a Yamcs installation, and is generally referred to as Yamcs Server. When Yamcs Server starts, it activates data links and starts listening for network connections from client programs.

11.3. yamcs-server init script

Yamcs distributions include an init script for starting and stopping Yamcs Server in System V-style. This script is located at /etc/init.d/yamcs-server and should not be run directly but instead via your system's service manager. This will perform proper stepdown to the yamcs user. For example: systemd start|stop|restart|status yamcs-server Or alternatively: service yamcs-server start|stop|restart|status The init script accepts these commands:

start

Starts Yamcs Server and stores the PID of the yamcsd process to /var/run/yamcs-server.pid

stop

Stops the Yamcs Server process based on the PID found in /var/run/yamcs-server.pid

restart

Stops Yamcs Server if it is running, then starts it again.

status

Checks if Yamcs Server is currently running. This does a PID check and will not detect a Yamcs Server runtime that has been started on the system without use of this init script.

Chapter 12. 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.

The web interface is separated in three different modules:

- *Monitor* provides typical monitoring capabilities (displays, events, …)
- *MDB* provides an overview of the Mission Database (parameters, containers, …)
- *System* provides administrative controls over Yamcs (tables, services, …).

All modules are aware of the privileges of the logged in user and will hide user interface elements that the user has no permission for. For normal operations access to the Monitor and MDB section should be sufficient.

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 redirect to the default page for that instance (i.e. the Display overview).

12.1. Monitor

The Monitor module within the Yamcs web interface provides typical operational views.

The Monitor Module is always visited for a specific Yamcs instance and processor. Every view has in the top right corner an indicator that shows the current processor and that shows the time for that processor. From this widget, you can choose to start a replay of past data. When that happens, you will switch to this replay processor and would see the widget reflecting the replay time.

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	Planned	No plans to support (use Yamcs Studio)
uss	USS displays	Advanced support	No plans to support (use USS Editor)
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.

Displays may be visualized in full screen, which helps remove distractions. This will not scale the display, so make use of the zoom in/out buttons before going full screen if you would like your display to appear larger.

Layouts

A layout allows to combine multiple displays in a single view. This is particularly useful if your displays are small. Layouts are personal and are linked to your user account. Create a layout via the “Create Layout” button in the toolbar. Open any number of displays via the sidebar and organize them together. Click “Save” when you want to save the changes to your layout for later use.

Individual display frames can be zoomed in or out by dragging the corner in the right-bottom.

A layout may be visualized in full screen, which helps remove distractions.

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.

Alarms

Shows an overview of the current alarms. Alarms indicate parameters that are out of limits.

Commands

Shows the latest issued commands.

TM Archive

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.

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.

12.2. MDB

The MDB module within the Yamcs web interface provides a set of views on the Mission Database.

The Monitor 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.

Space Systems

Space systems represent the highest conceptual entity of the MDB. Space systems act like folders and may contain other space systems, parameters, containers and others.

The Space Systems view shows a flat list of all space systems in the MDB. By clicking on one of the entires, you can navigate to a detail view for the space system which shows the parameters, containers, commands and algorithms that directly belong to that space system.

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.

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.

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.

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.

12.3. System

The System module within the Yamcs web interface provides a set of administrative views on Yamcs.

The System Module is always visited for a specific Yamcs instance, although some of the information may be global to Yamcs.

Dashboard

This provides a quick glance at the running system. It shows a quick graph of JVM system parameters and provides some basic server information such as the version number of Yamcs.

Links

Shows a live view of the data links for this instance. Link can be managed directly from this page.

Services

Shows the services for this instance. The lifecycle of these services can be managed directly from this page.

Processors

Shows a live view of the processors for this instance. This includes both persistent and non-persistent processors. Each processor has a detail page that allows seeing some statistics on the incoming packets, and that provides management controls over the command queues for this processor.

Clients

Shows the clients connected to this instance.

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.

Streams

Shows the streams for this instance. Each stream has a detail page that shows details about its SQL definition.