Yamcs Server Manual

October, 1st 2015

www.yamcs.org

Yamcs Server Manual

YAMCS-SA-MA-001

This version was published October, 1st 2015
A later version of this document may be available at www.yamcs.org

Table of Contents

Chapter 1. Introduction

1.1. Fundamentals

The Mission Database is a dictionary containing the description of all the telemetry packets, parameters and commands.

A Yamcs instance is used to provide a separation between different processing domains (basically different Mission Databases). Each instance has its own archive.

Streams are used to transfer data inside components of the same instance, running in the same Java Virtual Machine.

Processors are connection points for several different streams of the same instance (typically at least one TM and one TC) and processing data according to a Mission Database. Multiple processors can exist in the same instance, processing data for different time periods.

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

Data Links represent special components that inject data coming from external systems.

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 sourcecode. In the future it might be

externalised in configuration files to allow a certain degree of customisation.

Instances

The Yamcs instances provide means for one Yamcs server to monitor/control different payloads or version of the payloads 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 a 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 “read only” mode by disabling the components that inject data.

Streams

Streams are inspired from the domain of Complex Event Processing (CEP) or Stream Processing. They are similar to database tables, but they ‘contain’ continuously moving data. SQL-like statements can be defined on streams for filtering, aggregation, merging or other operations. Yamcs uses the streams for distributing data between all the 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 adding HornetQ wrappers. A wrapper is a HornetQ address that is configured such that each message published to the address gets transferred to the stream and each tuple that comes via the stream is transformed into a HornetQ message and published via the address. An external client then can bind a queue to that address and receive all the messages that transit through the respective stream. Unlike the Yamcs streams which are synchronous and very light, the HornetQ addresses are asynchronous and involve more overhead.

Currently the only way to see all the streams that are running inside a Yamcs server, is using a JMX client like JConsole. JConsole can also be used to inspect the status of the HornetQ addresses.

Processors

Mission Control Systems like Yamcs process TM/TC according to the Mission Database definitions. Yamcs supports concurrent processing of parallel streams; one processing context is called *Processors*. 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 processor each time a parameter retrieval is requested.

Data Links

Data Links represent special components that communicate with the external world. There are three types of Data Links: TM (called TM Providers), TC (called TC Uplinkers) and PP (called PP Providers). 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. Note that any stream (like the realtime TM stream) can be linked to a HornetQ address, making it possible to inject data externally. However, the Data Links can report on their status and can also be controlled by an operator to connect/disconnect to/from the data sources.

1.2. Installation

Dependencies

OS	Linux or Windows, 32bit or 64bit
Hardware	RAM >= 1Gb, HD >= 500Gb (dependent on amount of data archived)
Java runtime (JRE)	>= version 1.8
Tokyocabinet	>= version 1.4
libjtokyocabinet	>= version 1.22

Installation

Yamcs is delivered as an rpm (or deb) package and installation is achieved using the rpm command:

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

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

- `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`.

In addition to the core 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-busoc
  Provides the SOLAR (ISS/Columbus payload) event decoder and a few SOLAR specific derived variables.

- yamcs-erasmus
  Provides the EDR and FASTER (also ISS/Columbus payloads/instruments) event decoder and some specific derived variables.

The extensions are not part of the Yamcs open-source release. They only make sense in the specific USOC environment.

Configuration

The Yamcs configuration files are encoded using the yaml format. This format allows to encode in a human readable way (unlike XML) the most common data types: numbers, strings, lists and maps. For detailed syntax rules, please see http://www.yaml.org.

The starting configuration file is `yamcs.yaml`. It contains a list of Yamcs instances. For each instance, a file called `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.

The logging level is configured in `logging.yamcs-server.properties`. This file is used to configure the standard Java logging framework, and is encoded in standard java properties format. The formatting of the java properties files is described here: http://docs.oracle.com/javase/6/docs/api/java/util/Properties.html#load(java.io.Reader).

Updating

Upgrading is done using the rpm command:

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

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 version and to implement the newly added options into the old configuration files.

Removing

Yamcs Server can be removed (erased) using the rpm command:

```
rpm -e yamcs
```

Starting the Yamcs Server

Normally the 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/yamcs-server.sh`, but use of the `service` command is preferred.

Regardless of how Yamcs server is started, all the options are read from the configuration file `yamcs.yaml`.

Chapter 2. Data Links

2.1. TM Providers

TM Providers are components that collect data from external sources and inject them into a Yamcs stream. They are defined in the instance configuration file as part of the `tmProviders` list.

Each TM Provider is defined in terms of a name (which allows identifying it in the Yamcs Monitor), a class (the java class instantiated by Yamcs to load the provider), a specification (used as an argument when instantiating the class) and the name of the stream where the data will be injected. 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 startup.

DassPacketProvider

Receives telemetry packets (PathTM) from DaSS.

At startup the DassPacketProvider loads configuration properties defined in the configuration file `dass.yaml`. It builds the list of packet specifications which it should subscribe to DaSS. A specification is an object (vehicle id, packet type, apid, private header source). The list is built as follows:

- If the property `tmSubscriptionList` is defined, then it considers only the specifications matching this property.
- If the property `tmSubscriptionList` is not defined, then all the APIDs defined in the MDB are considered as part of the subscription list with the vehicle id set to 2 (=Columbus), the packet type set to 1 (=payload) and the private header source set to -1 (=all). Please note that this kind of subscription only works when connected to the DaSS kernel not when connected to the USOC router (the USOC router does not support the wildcard subscriptions).

Once the subscription list is built, the DassPacketProvider regularly tries to subscribe all the unsubscribed specifications from the list. At any time the Yamcs Monitor can be used to inspect the status of the subscription list. The Yamcs Monitor can also be used to close/re-open the connection to DaSS.

Secure Connections and Certificates

If the property `certificate` is specified, it has to point to a certificate file which will be then used by the DassPacketProvider (also the DassPpProvider and the DassTcUplinker) to connect to DaSS (or to the USOC router) in an encrypted way. Unlike the TMR in CD-MCS, the DassPacketProvider will try to verify the certificate of the server (using java built-in mechanism). Normally the Yamcs Server is started (by the script `yamcs-server.sh`) with the additional flag:

```
-Djavax.net.ssl.trustStore=yamcs_root_directory/etc/trustStore
```

This option configures java to trust the certificates signed with the certificates stored in the trustStore file.

For the secure connection to work, the file `etc/trustStore` has to be populated with the key

which has signed the DaSS server key. This can be easily done using the command *keytool* delivered as part of the Java distribution:

```
keystore -import -alias esa_root -keystore etc/trustStore -file
esa_root.crt
```

This command will import the file `esa_root.crt` into the java key store.

TcpTmProvider

Provides packets received via plain TCP sockets. The packets in CCSDS format are expected one after the other without any delimiter or separator (the length is deduced from the CCSDS header).

The specification consists of a name which selects a configuration defined in the file `tcp.yaml`.

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

TmapTmProvider

This provider is very similar with the TcpTmProvider above except that it expects packets having an extra 32 bytes PaCTS header followed by the CCSDS header. The PaCTS header is discarded.

MulticastTmProvider

This provider listens to a multicast (or UDP) port for datagrams containing CCSDS packets.

The specification consists of a name which selects a configuration defined in the file `multicast.yaml`

2.2. PP Providers

PP Providers are components that collect processed parameters (PP) from external sources and inject them into a Yamcs stream. They are defined in the instance configuration file as part of the `ppProviders` list.

Each PP Provider is defined in terms of a name (which allows identifying it in the Yamcs Monitor), a class (the java class instantiated by Yamcs to load the provider), a specification (used as an argument when instantiating the class) and the name of the stream where the data will be injected. There is also a property `enabledAtStartup` which allows to enable (default) or disable the PP provider for connecting to the external data source at the server startup.

DasPpProvider

Implements processed parameters from DaSS.

At startup the DassPpProvider loads configuration properties defined in the configuration file `etc/dass.yaml`. It builds the list of processed parameters which have to be subscribed from the MDB taking all the UMI tables found in the configured CCU.

Once the list of processed parameters is built, it tries regularly to subscribe to any unsubscribed parameter. Using the Yamcs Monitor it is possible to stop the subscription to DaSS and to close the connection.

The processed parameters are made available to the subscribing clients, using names like opsname_PP where opsname is the original name of the processed parameter (i.e. name at the source).

Unlike CD-MCS the processed parameters are made available via Yamcs with the telemetry status received from DaSS. In CD-MCS the other attributes of the processed parameters are made available via other parameters like opsname_ST, etc. In Yamcs these are not initialized, even the subscription is refused.

The mapping between DaSS status and Yamcs parameters is documented in section .

MulticastPpProvider

Implements processed parameters received via multicast from the TMR

At startup the DassPpProvider loads configuration properties defined in the configuration file `etc/multicast.yaml` (see). It builds the list of processed parameters which have to be considered from the MDB taking all the UMI tables found in the configured CCU.

The processed parameters are sent by TMR packetized in packets of variable size. Each packet is received in a UDP datagram.

The MulticastPpProvider uses some classes from the DaSS API to decode these packets which are then made available to the clients using the same mechanism and the same DaSS to CIS mapping like the DassPpProvider. The parameters which are not defined in the UMI maps loaded

from the MDB are discarded.

Using the Yamcs Monitor the processing of packets can be enabled/disabled. In addition the MulticastPpProvider collects simple statistics with the number of UDP datagrams received and the number of processed parameters in the last datagram. These statistics can also be seen using the Yamcs Monitor.

SimulationPpProvider

Some tests request data to be simulated. This can be achieved by using the SimulationPpprovider. This simulation pp provider uses in input scenarios that are defined in XML files.

The XML scenario file allows to describe the parameters sent, their generation time, acquisition time, engineering value and monitoring value. Parameters are organized in ‘sequence’ that can be repeated to allow more complex scenarios. The speed of the simulation can be defined, by setting the duration of a simulation step

The format of XML scenario file is defined in annex XXX

A user interface is available to visualize a library of scenario and select the scenario that YAMCS should play. This tool is documented in section XXX

2.3. TC Uplinkers

DassTcUplinker

Sends telecommands to the DaSS. It supports decoding of acknowledgments from DaSS, MCS, FSC, FRC, A, B, and C and upon reception of any of them, it generates two command history entries:

`<source>_Status`	"OK" or "NACK msg"
`<source>_Time`	the time at the reception of the acknowledgment

Where `<source>` is one of:

- Acknowledge_DaSS
- Acknowledge_MCS
- Acknowledge_FSC
- Acknowledge_FRC
- Acknowledge_A
- Acknowledge_B
- Acknowledge_C

The uplinker also supports resequencing events which is a process through which one upstream commanding server can change the sequence count of the uplinked telecommand, in order to preserve the correct sequence across multiple commanding clients.

TcpTcUplinker

Sends telecommands via TCP. Can be connected directly to the Columbus Emulator. It emulates all the DaSS related acknowledgments presented in the section above, with ACK OK.

TcapTcUplinker

Sends telecommands to the TCAP process which is a part of PaCTS.

TCAP is doing its own CCSDS APID sequence counting and the protocol does not foresee a way to pass this information back to the TcapTcUplinker. Thus TcapTcUplinker will have no idea on what the final sequence count will be and will not generate corresponding re-sequencing command history events. Thus the Command History will not be able to associate the corresponding CCSDS command response (sent by the BSW) to the telecommand sent.

Chapter 3. Mission Database

3.1. TM Loaders

CD-MCS MDB

This loader loads the telemetry 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 testConfigurationNote that 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 and to 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 can not 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).

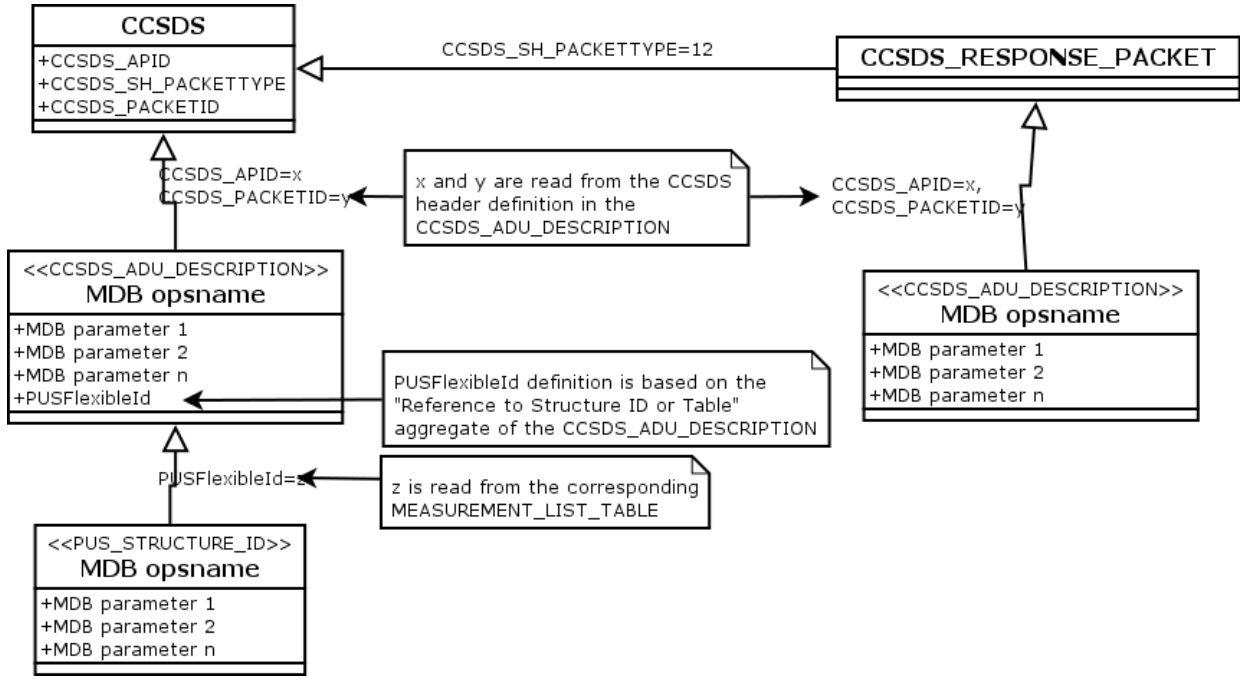
The packet description in CD-MCS MDB is spread over different structures. When read into Yamcs, they are converted into the XTCE structures as follows:

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.



Excel Spreadsheet

This loader constructs the MDB using containers and parameters defined in one or more Excel spreadsheets.

The loader is configured in `etc/mdb.yaml` (see for a sample configuration). Specify the ‘type’ as `sheet`, and provide the location of the spreadsheet file in the spec attribute. Additional spreadsheets may be specified in a `subLoaders` list, using the type and spec attributes for each additional spreadsheet.

The format of the Excel spreadsheet is described in .

XTCE files

This loader reads an MDB saved in XML format compliant with the XTCE specification. For more information about XTCE, see .</para>

The loader is configured in `etc/mdb.yaml` (see for a sample configuration). Specify the 'type' as `xtce`, and provide the location of the XML file in the spec attribute.

3.2. PP Loaders

Processed Parameters represent parameters that are processed by systems outside Yamcs. Currently, the only such system supported is DaSS. Yamcs simply has to know the name of parameter and does not do any extra check (like out of limits, validity, etc). In addition, each parameter can be part of a group (which is just a string). The parameters part of the same group are stored together and they appear as such in the Yamcs Archive Browser.

MDB PP Loader

The MDB PP loader scans a configured CD-MCS MDB (using direct Oracle connection) 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.

Flat-file PP Loader

The flat-file PP loader reads list of parameter names, groups and opsnames from a tab-separated file.

3.3. TC Loaders

Currently the only TC loader is the CD-MCS MDB which reads data directly from the CD-MCS Oracle.

Chapter 4. Telemetry Processing

4.1. 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 toand . This sections details 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, then 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

4.2. 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. One out of:

null	WATCH	WARNING	DISTRESS	CRITICAL	SE
DISABLED	WATCH_LOW	WARNING_LOW	DISTRESS_LOW	CRITICAL_LOW	SE
IN_LIMITS	WATCH_HIGH	WARNING_HIGH	DISTRESS_HIGH	CRITICAL_HIGH	SE

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 accross 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's 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).

4.3. Alarms

Yamcs supports the XTCE notion of *alarms*. Based on the value of a parameter, Yamcs assigns a so-called 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_LOW
- WATCH_HIGH
- WARNING_LOW
- WARNING_HIGH
- DISTRESS_LOW
- DISTRESS_HIGH
- CRITICAL_LOW
- CRITICAL_HIGH
- SEVERE_LOW
- SEVERE_HIGH

The additional LOW/HIGH suffix indicates whether the parameter 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’).

4.4. Commanding

Yamcs contains an HLCL parser capable to parse the telecommanding requests coming from the MCS Tools.

XTCE structures are not supported for commanding, and neither are command preconditions or postconditions (i.e. checking that telemetry parameters have certain values before or after sending the command).

Command Queues

In Yamcs when a command is sent by the CIS client it doesn't go directly to the TcUplinker but instead it goes 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 telecommand, the command will be rejected and not appear at all in the queue. In fact, the command is already rejected when the prepareCommand CORBA call is made. This means that a user will not be able to open using the MCS Tools a stack containing a command for which he does not have authorization.

The available queues are defined in the file `etc/commandqueue.yaml` (see).

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 role is specified. 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 whose role is not defined by any other queue (not recommended), the command will be put in the default queue. The default queue can be redefined in the `command-queue.yaml` in order to have a different state.

The content of the command queue can be inspected using the Yamcs Monitor as explained . To be able to control the Command Queues, the user needs the MayControlCommandQueue privilege.

Chapter 5. Security

5.1. Authentication

Yamcs allows both secure (SSL based) and non-secure connections. The level of authentication performed is directly related to the connection type:

- Non-secure connections: the GSSUP mechanism part of CSIv2 CORBA specifications is used to obtain information about username. Although a password is carried via this mechanism, it is not checked against anything to avoid the users being required to enter their password each time they start a Monitoring or Commanding Tool. For this reason, non-secure connections should only be allowed from trusted environments.
- Secure connections: these are based on SSL. The SSL connections require client authentication and the LDAP database is searched for the username based on the certificate. The certificate has to be placed in the LDAP attribute `userCertificate`.

Asserted Identities

The CSIv2 CORBA specifications include a mechanism through which a user (typically corresponding to a proxy) can execute CORBA calls on behalf of other users. The mechanism is called identity assertion: the proxyuser asserts the identity of the real user.

The Yamcs Server supports the identity assertion mechanism, imposing at the same time some restrictions meant to improve security:

- Identity assertion can be made only over secure (SSL) connections. As mentioned above the client always has to authenticate over secure connections and the proxyuser LDAP entry has to define the proxyuser certificate.
- The LDAP entry for the user that asserts other identities (typically `cn=proxyuser,ou=People,o=usoc`) has to contain a subnode `cn=assertedIdentities` of type `groupOfNames` which has `member` attributes pointing to all the users whose identities can be asserted. If the proxyuser attempts to assert an identity not in the list, a NO_PERMISSION error will be thrown.

5.2. Authorization

Yamcs implements four types of privileges: System Privileges, TC Privileges, TM Sequence Containers (TM Packets) Privileges and TM Parameter Privileges.

The privileges are assigned to users through the use of roles: a user has specific roles, and some role is required for a specific privilege. If there is a match between the role assigned to the user and the role required for a privilege, then the user is allowed to pass the restriction.

All the user, role and privilege definitions are looked up in the LDAP database. Yamcs reads only LDAP objects of type `groupOfNames`. The access to the LDAP server is done using the properties from the `privileges.yaml` (see). Yamcs requires read-only access to the LDAP.

The algorithm used by Yamcs to check if the user has a privilege (e.g. `EUTEF_Tlm_Pkt_HK_DHPU`) is as follows:

- From the path configured by `privilege.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=corba_username` from the `privilege.userPath`.
- For each role found previously (e.g. `TRBIOLAB-Operator`), 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 (e.g. `EUTEF_Tlm_.*` and `EUTEF_TRIBOLAB_.*`).
- Each item from the list of privileges that the user has (e.g. `EUTEF_Tlm_.*`) is considered as a regular expression and it is matched with the privilege that is required for the given operation (e.g. `EUTEF_Tlm_Pkt_HK_DHPU`). Note that the regular expression matching has been introduced in order to avoid multiplying the entries for TM/TC information. It can also be used for the system privileges (e.g. creating a entry `May.*` will allow everything) but it is not recommended.

The information found using the algorithm above is cached for 30 seconds such that when the user opens a USS display or a command stack containing many items, it is not necesary to repeat the same LDAP queries many times. The side effect, of course, is that a change in the LDAP database can take 30 seconds to be noticed by Yamcs.

System Privileges

Used to impose general limits, such as the privilege to command, privilege to control the channels with the Yamcs Monitor, etc. The following privileges are supported:

MayCommandPayload	This privilege is prerequisite for sending any command (in addition to that required for the command itself). Without this privilege, the user will not even be able to subscribe to commanding via the CORBA call subscribeCommanding. The privilege is checked each time the user prepares or sends a command.
	This privilege allows to modify the command history with

MayModifyCommandHistory	additional command history events. It is normally required only by the command history application. The privilege is checked each time the user tries to add a command history event.
MayControlCommandQueue	This privilege is required in order to be able to inspect and control the command queue. See . The privilege is checked each time the user calls one of the operations for controlling the command queues.
MayControlChannels	This privilege allows the user to control channels (e.g. enable/disable TM/TC, connect/disconnect sessions) other than their own. The channel control is done using the Yamcs Monitor.
MayControlArchiving	This privilege allows to enable archiving for the channels created on the fly.

TC Privileges

In addition to the MayCommandPayload, each telecommand has an implicit privilege required to send the telecommand. The privilege has the same name as the opsname of the telecommand.

The TC privilege is checked each time when the user tries to send a command.

TM Sequence Containers Privileges

Similar to the TC, each TM Sequence Container has an implicit privilege required to monitor it. Note that due to the inheritance nature of the Sequence Containers, giving access to a higher level container, implicitly allows access to all the inherited containers. For example if a user has the privilege to monitor the CCSDS container which is the root container, he will get all the CCSDS packets even though he may not have explicit access to the EUTEF_Tlm_HK_DHPU container.

The TM Sequence Container Privileges are checked only at subscription time. Once the user is subscribed to a sequence container, changing the privileges in the LDAP database will have no effect on the ongoing subscription.

TM Parameters Privileges

Similar to the TC and to the TM Sequence Containers, each TM parameter has associated an implicit privilege required to monitor it.

Similar to the TM Sequence Containers, the privileges are checked against the LDAP database only at subscription time.