Yamcs Server Manual

YAMCS-SA-MA-001

October, 12th
2015

www.yamcs.org

Yamcs Server Manual

YAMCS-SA-MA-001

This version was published October, 12th 2015
A later version of this document may be available at www.yamcs.org

Table of Contents

1. Introduction **2**
Fundamentals 2
Installation 4
Acronyms 6

2. Data Links **8**
TM Providers 8
PP Providers 10
TC Uplinkers 12

3. Mission Database **13**
TM Loaders 13
PP Loaders 16
TC Loaders 17
Excel Specification 18

4. Telemetry Processing **31**
Packet Telemetry 31
Algorithms 33
Alarms 35
Commanding 36

5. Security **37**
Authentication 37
Authorization 38

6. Configuration **41**
command-queue.yaml 41
logging.yamcs-server.properties 42
mdb.yaml 43
multicast.yaml 44
privileges.yaml 45
tcp.yaml 46

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 source code. 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 represent 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 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 *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 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 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-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 `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.

The logging level is configured in etc/logging.yamcs-server.properties.

Upgrading

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 versions 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 `etc/yamcs.yaml`.

1.3. Acronyms

Acronym	Definition
ACK	Acknowledgement
ADU	Application Data Unit
API	Application Programming Interface
APID	Application ID
BSW	Basic Software
CCSDS	Consultative Committee for Space Data Systems
CCU	Configuration Control Unit
CD-MCS	Columbus Decentralised Mission Control System
CDU	Configuration Data Unit
CEP	Complex Event Processing
CIS	CORBA Interface Servant
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSI	Common Secure Interoperability Protocol Version 2
DaSS	Data Services Subsystem
EDR	European Drawer Rack
EGSE	Electrical Ground Support Equipment
EUTEF	European Technology Exposure Facility
FASTER	Facility for Adsorption and Surface Tension
FRC	Facility Responsible Centre
FSC	Facility Support Center
GSSUP	Generic Security Services Username Password
HK	House Keeping
HLCL	High Level Command Language
ID	Identifier
ISS	International Space Station
JMX	Java Management Extensions
JRE	Java Runtime Environment
JSR	Java Specification Requests
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MCS	Monitoring and Control Subsystem
MDB	Mission Data Base
NACK	Not Acknowledged
PaCTS	Payload Computer Test System
PP	Process Parameter

Acronym	Definition
RAM	Random Access Memory
RESP	Response
SCOE	Special Checkout Equipment
SH	Secondary Header
SID	Short Identifier
SQL	Structured Query Language
SSL	Secured Socket Layer
TC	Telecommand
TCAP	Transaction Capabilities Application Part
TCP	Transmission Control Protocol
TM	Telemetry
TMR	Telemetry Router
TOC	Table of Content
TRIBOLAB	Tribology Laboratory
UDP	User Datagram Protocol
UHB	User Home Base
UMI	Universal Measurement Identifier
USOC	User Support and Operations Center
USS	Unified Synoptic System
XML	Extensible Markup Language
XTCE	XML Telemetry & Command Exchange

Chapter 2. Data Links

Data Links represent input or output flows to Yamcs. 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 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.

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

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 an identifying name, 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 start-up.

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 etc/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 etc/multicast.yaml.

DassPacketProvider

Receives telemetry packets (PathTM) from DaSS.

A t start-up the DassPacketProvider loads configuration properties defined in the configuration file

`etc/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.

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.

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 an identifying name, 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 start-up.

DassPpProvider

Implements processed parameters from DaSS.

At start-up 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.

MulticastPpProvider

Implements processed parameters received via multicast from the TMR

At start-up the MulticastPpProvider loads configuration properties defined in the configuration file etc/multicast.yaml. 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 Yamcs Studio or 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. The SimulationPpProvider takes as 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 a 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.

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

The Yamcs Mission Database is composed of the following parts, contained in an XTCE hierarchical structure:

- Telemetry
- Telecommands
- Processed 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 database source type.

3.1. TM Loaders

Excel Spreadsheet

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

The loader is configured in etc/mdb.yaml. 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.

XTCE files

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. Specify the 'type' as `xtce`, and provide the location of the XML file in the `spec` attribute.

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 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:

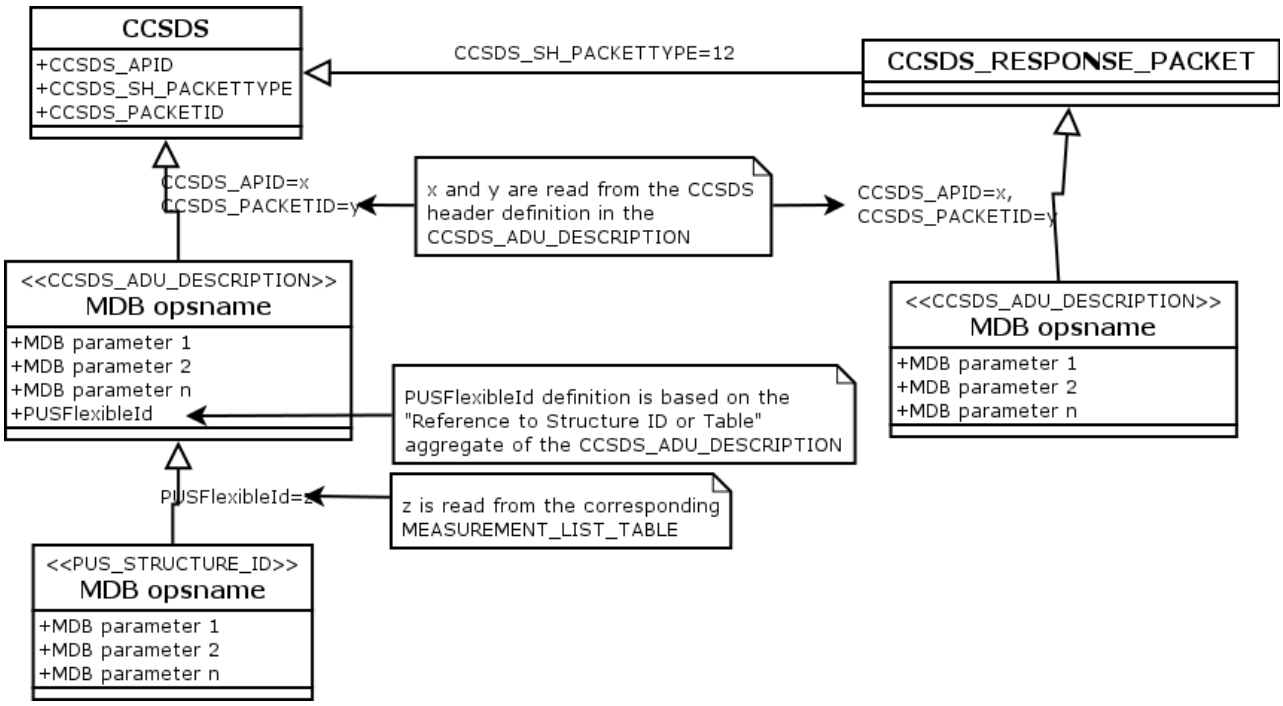
`CCSDS_APID`	the APID in the CCSDS primary header
`CCSDS_SH_PACKETTYPE`	the packet type in the CCSDS secondary header
`CCSDS_PACKETID`	the packet type in the CCSDS secondary header

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`.


CCSDS
+CCSDS_APID
+CCSDS_SH_PACKETTYPE
+CCSDS_PACKETID
CCSDS_SH_PACKETTYPE=12
CCSDS_RESPONSE_PACKET
CCSDS_APID=x
CCSDS_PACKETID=y
x and y are read from the CCSDS header definition in the CCSDS_ADU_DESCRIPTION
CCSDS_APID=x,
CCSDS_PACKETID=y
<<CCSDS_ADU_DESCRIPTION>>
MDB opsname
+MDB parameter 1
+MDB parameter 2
+MDB parameter n
+PUSFlexibleId
<<CCSDS_ADU_DESCRIPTION>>
MDB opsname
+MDB parameter 1
+MDB parameter 2
+MDB parameter n
PUSFlexibleId definition is based on the "Reference to Structure ID or Table" aggregate of the CCSDS_ADU_DESCRIPTION
PUSFlexibleId=z
z is read from the corresponding MEASUREMENT_LIST_TABLE
<<PUS_STRUCTURE_ID>>
MDB opsname
+MDB parameter 1
+MDB parameter 2
+MDB parameter n

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.

Excel Spreadsheet

This the Yamcs Excel spreadsheets loader, which can define a full MDB based on the XTCE components: Telemetry, Telecommand, Process Parameters and Algorithms.

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

Excel Spreadsheet

This the Yamcs Excel spreadsheets loader, which can define a full MDB based on the XTCE components: Telemetry, Telecommand, Process Parameters and Algorithms.

CD-MCS MDB

The CD-MCS MDB reads data directly from the CD-MCS Oracle.

3.4. Excel Specification

Yamcs offers various ways of defining TM packets, parameters, commands and other mission items. The recommended way is to use a spreadsheet. This offers a good mixture between easy user manipulation and flexibility. This section explains the general and structural conventions that apply when defining a space system using Excel.

Conventions

- If specified, every opsname will automatically be prepended with the `opsname prefix` from the 'General' sheet
- The actual descriptive part of opsnames is WordCapitalized (eg. `HK_WatchdogError`)
- 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.

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
opsname prefix	Optional value which will automatically be prepended before every opsname, useful if all the parameters have a common system prefix

Containers Sheet

This sheet must be named "Containers", and the columns described must not be reordered. The sheet contains packets information, including their measurements. 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

container name	The opsname of the packet
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`
flags	Used to encode special cases. Flags are case-insensitive and can be concatenated together. • `I` – ignore completely when generating the Oracle MDB (In Yamcs the parameter is *not* ignored) • `D` – derived value, in this case the measurement must not have a calibration • `L` – little endian value
parameter name	The opsnames of the Parameters contained within this container, one per line after the line defining the opsname. Parameters are defined in the Parameters sheet. Note that the first parameter of a packet may not be on the same line as the packet opsname. Extension: for the SOLAR Instruments, this column has been extended to mean not only a measurement but also another packet in case the extension mechanism specified above is not enough.
relpos	Relative position (offset) of a measurement to its predecessor. A relpos of 1 means no gap between two measurements. A relpos of 9 means a gap of one byte (i.e. 8 irrelevant bits).
size in bits	The size of the container in bits. This is not usually needed but if the container is part of another container through aggregation, this size is used to determine the position of the next entry in the master container.
expected interval	If the container is not updated in this interval, all the parameters from it are marked as expired
description	Optional human-readable text
namespace:MDB:Pathname	Optional MDB alias for the packet

Parameters Sheet

This sheet must be named “Parameters”, and the columns described must not be reordered. The sheet contains measurements information (both for TM and RESP packets), excluding their calibration.

opsname	The opsname of the measurements. Any entry starting with `#` is treated as a comment row
bitlength	Length of the measurement, in bits, not needed for terminatedstring raw types
raw type	See 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:MDB:Pathname	Optional MDB alias for the packet

Raw Types

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

Type	Description
`uint`	Unsigned integer, need to specify bit length
`float`	Floating point number, need to specify bit length
`int`	Synonymous to `int(2s)`
`int(2s)`	Signed integer using 2's complement binary representation, need to specify bit length
`int(si)`	Signed integer using sign-magnitude binary representation, need to specify bit length
`boolean`	Boolean value of 1 bit. Value 1 is true and 0 is false. The bit length must not be defined as it is assumed to be 1
`bytestream`	Catch-all for any binary data, need to specify bit length
`string`	Synonymous to `TerminatedString(0x0)`
`TerminatedString(0x0)`	A string terminated by the specified byte.
`FixedString`	Fixed length string, specify size in bitlength
`PrependedSizeString`	Has size specified in the leading byte of the value

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 integer	From type `uint` or `string`
`int`	Signed integer	From `int`, `uint` or `string`
`string`	String	From `string`
`float`	Floating point number	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. If the parameter's enumerated value is unknown, the state value `UNDEF` is used instead.	From `int` or `uint`. A Calibrator is required.
`boolean`	A binary true/false value	From any raw type Values equal to zero, all-zero bytes or an empty string are considered *falsy*
`binary`	Catch-all for any binary data	From `bytestream` only

Derived Parameters Sheet

This sheet must be named “DerivedParameters”, and the columns described must not be reordered.
Derived Parameters are parameters whose values are output of Algorithms (defined in section Algorithm).

opsname	The opsname of the parameter. Any entry starting with `#` is treated as a comment row
bitlength	Length of the measurement, in bits, not needed for terminatedstring raw types
raw type	See 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:MDB:Pathname	Optional MDB alias for the packet

Local Parameters Sheet

This sheet must be named “LocalParameters”, and the columns described must not be reordered.
Local parameters are equivalent to Parameters but can be written by Yamcs clients.

opsname	The opsname of the parameter. Any entry starting with `#` is treated as a comment row
bitlength	Length of the measurement, in bits, not needed for terminatedstring raw types
raw type	See 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:MDB:Pathname	Optional MDB alias for the packet

Calibration Sheet

This sheet must be named “Calibration”, and the columns described must not be reordered. The sheet contains calibration data including enumerations

parameter name	Name of the calibration.
type	• `polynomial` for polynomial calibration • `pointpair` for pointpair calibration • `identical` for discrete calibration • `enumeration` for mapping enumeration states
calib1	• If the type is `polynomial`: coefficient • If the type is `pointpair`: start point • If the type is `identical`: engineering value • If the type is `enumeration`: numeric value
calib2	• If the type is `polynomial`: leave *empty* • If the type is `pointpair`: stop point corresponding to the start point in `calib1` • If the type is `identical`: engineering value corresponding to the raw value in `calib1` • If the type is `enumeration`: text state corresponding to the numeric value in `calib1`

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.

algorithm name	The identifying name of the algorithm. Any entry starting with `#` is treated as a comment row
text	The code of the algorithm
trigger	Optionally specify when the algorithm should trigger. If left blank, algorithms will trigger as soon as at least one of their input parameters is updated, which is expected to suit most use cases. Other options are: • `OnParameterUpdate('/some-param', 'some-other-param')` Execute the algorithm whenever *any* of the specified parameters are updated • `OnPeriodicRate(<fireRate>)` Execute the algorithm every `fireRate` milliseconds
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 name	Reference name of a parameter. Input parameters can come from any sort of parameter provider. Output parameters should be defined in the same spreadsheet as the algorithm definition. Both input and output parameters must be defined in the Parameters sheet and can have calibrations linked to them. In most cases there is no need to specify a bitsize for an output parameter. 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.

Example Definition

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

When monitoring results change, events are generated and can be followed in the Yamcs Event Viewer.

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 opsname 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.
argAssignement	name1=value1;name2=value2.. where name1,name2.. are the names of arguments which are assigned when the inheritance takes place
flags	for comands: A=abstract for argument: L = little endian
argument name	from this column on, most of the parameters are valid for arguments only which have to come starting next row after the command. The exceptions are: - - size in bits - description - aliases
relpos	relative position to the previous argument default is 0
size in bits	size in bits of the raw value
eng type	engineering type; can be one of: - uint - int - float - string - binary - enumerated - boolean - FixedValue FixedValue is like binary but is not considered an argument but just a value to fill in the packet.
raw type	raw type: can be one of uint, int, float or binary
(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

Command Options Sheet

This sheet must be named “CommandOptions”, and the columns described must not be reordered.
This sheet defines the options that can be applied to commands.

Command name	The opsname 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.

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.

Legacy Sheets

Legacy sheets remain supported, but are generally not used in new projects anymore.

MDB Sheet

This sheet is optional, but if present must be named “MDB”. The columns described must not be reordered.

version issue revision testversion	Used for CDU version tracking
instance	Organisation
systemtree	Version of the system tree used
cupath	Path in MDB
apid	APID used for TCs
element_configuration	Root system
mission CDU version CDU description	Inserted into generated CDU

TC Sheet

This sheet must be named “TC”, and the columns described must not be reordered. The sheet contains telecommand descriptions and their parameters.

Constant parameters (i.e. cannot be chosen when building the command) are specified as constants (not a range, not an enum, only a lowlimit/value) and have no name.

Ordinary parameters (parameters to be filled in by the user of the TC) must have a name and an enum or a range.

Parameters that are calculated (eg. checksum) have a range or an enum but no name.

packet opsname	The opsname of the TC
packet partial full pathname	The part of the full pathname, relative to the CDU (maintained by SpaceApps)
packetid	Identifying number
response packet opsname	The opsname of the response packet, can be left empty if no response packet is expected
parameter name	Each parameter is optionally given a name
flags	Used to encode special cases. Flags are case-insensitive and can be concatenated together. • `I` – ignore completely when generating the Oracle MDB (in Yamcs the parameter is not ignored). • `C` – calculated value (only for parameters)
bits	Specifies the number of bits the parameter has. Can be any non-zero number
datatype	A raw type: `uint`, `int`, `string`, `varstr`, `float` or `enum`. All case insensitive. `string` has a fixed size that is padded if necessary. `varstr` has a maximum size and will not be padded
rel position	Specifies the offset of this parameter to the last bit of the previous parameter. No empty bits means a relative position of 1. Three irrelevant bits between parameter n and parameter n+1 means a relative position of 4

Value/lowlimit	Contains engineering values. There are several entry possibilities:
	• the column contains 1 value, but the highlimit column is left empty for this parameter, the value means a fixed value for this parameters
	• the column contains 1 value and the highlimit column also contains a value for this parameter, the 2 values specify a range for this parameters
	• the column contains semicolons and each value seperated between semicolons (do not use spaces before/after the semicolons) is either:
	  ◦ a '=' b pair. Lefthandside = textual representation of the righthandside value, this textual representations cannot be longer than 8 characters
	  ◦ x where x is a number and meaning that any value larger or equal to x is also a valid value. Having this means that also a highlimit must be filled in combining the previous ones is also possible.
	
	For example, `ON=1;OFF=0;3` means that value 1 is encoded as `ON`, value 0 is encoded as `OFF` and any other value greater than or equal to 3 is also a valid value (if the highlimit field exists and is greater than 3)

highlimit	The upper limit of the values that the parameter can take
engunit	Free-form engineering units for the parameters
calibration	Name of a calibration described in the Calibration Sheet
description	An optional description for the commands or parameters.

RESP Sheet

This sheet is optional, but if present must be named “RESP” and the columns must not be reordered. The sheet specifies the basic identifying information for packets which are expected in response to TC.

response opsname	OpsName of the packet
response partial full pathname	The part of the full pathname, relative to the CDU (maintained by SpaceApps)
apid	APID to which the packet belongs
packetid	ID of the packet
measurement opsname	OpsName of the triggering packet
relpos	
description	Optional human-readable text

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 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

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	SEVE
DISABLED	WATCH_LOW	WARNING_LOW	DISTRESS_LOW	CRITICAL_LOW	SEVE
IN_LIMITS	WATCH_HIGH	WARNING_HIGH	DISTRESS_HIGH	CRITICAL_HIGH	SEVE

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).

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 supported for commanding, including command preconditions and 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/command-queue.yaml.

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 etc/command-queue.yaml in order to have a different state.

The content of the command queue can be inspected using Yamcs Studio or Yamcs Monitor. Control over the command queues, requires the `MayControlCommandQueue` privilege.

Chapter 5. Security

This chapter describes the authentication and authorization mechanisms implemented in the Yamcs Server. Please note that these are in effect only when privileges are enabled by setting `enabled: true` in etc/privileges.yaml.

If privileges are enabled, the Yamcs Server will connect to LDAP and read the user, roles and privileges from the directory paths specified in the configuration file above.

If privileges are disabled, the Yamcs Server will not even attempt to connect to LDAP and will assume that any user has all privileges required for the operations attempted.

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. 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. `TRIBOLAB-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.
`MayModifyCommandHistory`	This privilege allows to modify the command history with 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 command queues. 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 `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.

Chapter 6. Configuration

6.1. command-queue.yaml

```
# Definitions of command queues
# Eeach queue has a name, a default state and a list of roles
# There are three possibile 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 commandqueue called "default". It can be  redefined here
# in order to have a different state
# If a command comes from a user which is not defined by any other queue,
#   it is put into the default queue

queueNames: [ops, uhb]

ops:
    state: enabled
    significances: [none]
    roles: [Operator]

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

uhb:
    state: disabled
    roles: [UHB-Operator]
```

6.2. 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 at [http://docs.oracle.com/javase/6/docs/api/java/util/Properties.html#load(java.io.Reader)] (http://docs.oracle.com/javase/6/docs/api/java/util/Properties.html#load(java.io.Reader))

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

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

java.util.logging.ConsoleHandler.level = WARNING
java.util.logging.ConsoleHandler.formatter = org.yamcs.CompactFormatter


org.yamcs.level=FINE
```

6.3. mdb.yaml

```
refmdb:
   # Valid loaders are: sheet, xtce or fully qualified name of the class
   - type: "sheet"
     spec: "mdb/refmdb-ccsds.xls"
     subLoaders:
         - type: "sheet"
           spec: "mdb/refmdb-subsys1.xls"

simulator:
    # Configuration of the active loaders
    # Valid loaders are: sheet, xtce or fully qualified name of the class
    - type: "sheet"
      spec: "mdb/simulator-ccsds.xls"
      subLoaders:
           - type: "sheet"
             spec: "mdb/simulator-tmtc.xls"
```

6.4. multicast.yaml

```
#Properties for the MulticastPpProvider and MulticastTmProvider
example:
    ppGroup: localhost
    ppPort: 10100
    tmGroup: localhost
    tmPort: 10200
```

6.5. privileges.yaml

```
#configuration of the privileges
# if privileges.enabled is set to false, the connection to realm is not even
# attempted and hasPrivilege method always returns true
#
# if privileges.enables is set to true, you need to enable security of hornetQ and
ensure
# the hornetQ roles are assigned to user in the chosen realm.
enabled: false

#one of: LdapRealm, YamlRealm
realm: org.yamcs.security.YamlRealm

#maximum number of CIS sessions clients are allowed
maxNoSessions: 10

ldaphost:        host
userPath:        "ou=People,o=usoc"
rolePath:        "ou=Roles, ou=Operation, ou=Columbus,ou=Projects,o=usoc"
systemPath:      "ou=System, ou=yamcs, ou=Applications, ou=Operation,
ou=Columbus,ou=Projects,o=usoc"
tmParameterPath: "ou=TM_PARAMETER, ou=yamcs, ou=Applications, ou=Operation,
ou=Columbus,ou=Projects,o=usoc"
tmPacketPath:    "ou=TM_PACKET, ou=yamcs, ou=Applications, ou=Operation,
ou=Columbus,ou=Projects,o=usoc"
tcPath:          "ou=TC, ou=yamcs, ou=Applications, ou=Operation,
ou=Columbus,ou=Projects,o=usoc"

# yaml file, if using YamlRealm
yamlRealmFilename:  credentials.yaml
```

6.6. tcp.yaml

```
#configuration for the TcpTmProvider and TcpUplinker
# they look for properties like
#spec:
#    tmHost:
#    tmPort:
#    tcHost:
#    tcPort:
#    minimumTcPacketLength (by default 48)

local:
    tmHost: localhost
    tmPort: 10015
    tcHost: localhost
    tcPort: 10025
```