Core Services user guide

Release 1.4.1

Software and Controls Group

Dec 05, 2018

Contents

CHAPTER

ONE

INTRODUCTION

This documents describes the use of the GMT OCS Core Services. The OCS Architecture Documment [ref] describes the overall design of the OCS and provides a description of the function and architecture of the OCS Core Services. In short the Core Services are part of the OCS distributed architecture and provides services that the OCS distributed Components can use to perform their function. The following diagram shows the OCS Core Services in the context of the overall OCS.

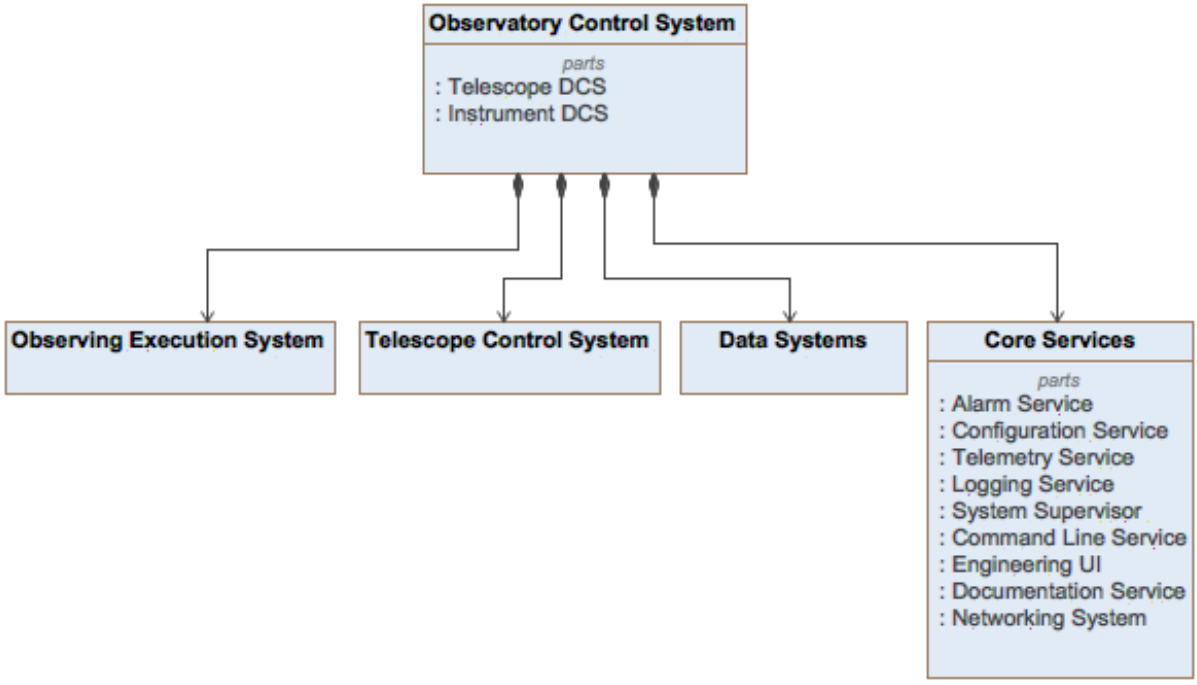

Fig. 1.1: Core Services in the context of the OCS

The Core Services that are available in the latest release are:

- Logging Service
- Alarm Service
- Telemetry Service
- Configuration Service
- Supervisory Service

Although the fuctions offered by each service are orthogonal to each other, all of them share similar design and operation principles. The implementatio of the services is distributed and consists of the following components:

- A **Service Adapter** that allows Components to access the interface of the Service. The Core Framework presents the Service Adapters to the the Component base class via dependency injection, making the Core Services available to every Component in the system.
- A **Service Server** (or Servers) that process the data relative to the specific service. All the Core Services transport and process data in the form of Events. The Service Server makes the Events persistent and allows other Components to subscribe to the Events. It is also capable of receiving querys and reply with the result of executing the query in the Service Database. The Service Server is instanciated as part of each Service Server Application.
- A **Service Database** that is stores each server data.
- A **Service Client** that allows to interact with the Server. The Service Client can be instaciated by a Component. A Service Client Application is provided that facilitates interacting with each Service from the command line.

CHAPTER

TWO

SERVICE USAGE

This section describes the usage of the different components of each service. As all the services support the same options and operate in a similar way, examples of the usage of different services are included in the descriptions. The following command line application are installed as part of the SDK:

Service	Application
Alarm	alarm_server
Alarm	alarm_client
Configuration	conf_server
Configuration	conf_client
Configuration	conf_send (alpha)
Logging	log_server
Logging	log_client
Supervision	sup_server
Supervision	sup_client
Telemetry	tele_server
Telemetry	tele_client

2.1 Service Server Application

Description

The `<service>_server` command line application starts the alarm service server. For the purposes of developing and testing software components with the SDK it is recommended to start the service with the default options or with the logging set to debug in case of troubleshooting.

```
λ alarm_server&

# -> will start the server with default options (recommended)

λ alarm_server --logging debug

# -> will start the server with default options and will output debug log messages␣
↪(recommended for troubleshooting)
```

The following section shows the execution of the help command and the available options.

```
λ alarm_server --help

Usage: alarm_server_app [options]
```

```
If called without options alarm_server_app will use default values for them

Options:

--srv_conf_file=<name>    name of the service server configuration file
--database=<name>         name of the service database
--collection=<name>       name of the service collection
--help                    prints help
--logging=<level>         logging level
--name=<name>             application name
--scope=<local|global>    scope to load the configuration file
--conf_file=<name>        name of the application configuration file
--ctnr_conf_file=<name>   name of the container configuration file
```

Options

`--srv_conf_file`

This option specifices the name of the Server configuration file to be loaded during the Server startup. The configuration file defines the port configuration of each server. The following example shows the configuration of the Alarm Server. Configurations are stored in $GMT_LOCAL/etc/conf

```
module.exports =
   properties:
      uri:          {name: "uri",          default_value: 'gmt://127.0.0.
↪1:12000/alarm_sys/alarm_srv_pkg/alarm_srv'}
      name:         {name: "name",         default_value: 'alarm_srv'}
      host:         {name: "host",         default_value: '127.0.0.1'}
      port:         {name: "port",         default_value: '12000'}
      scan_rate:    {name: "scan_rate",    default_value: 1}
      acl:          {name: "acl",          default_value: 'PRIVATE'}
      db_uri:       {name: "db_uri",       default_value: 'mongodb://127.0.0.
↪1:27017/'}
      collection:   {name: "collection",   default_value: 'alarms'}
      database:     {name: "database",     default_value: 'gmt_alarm_1'}

   state_vars:
      ops_state:  { name: 'ops_state',    default_value: 'OFF'  }

   inputs:
      conf_in:          { name: 'conf_in',          port_type: 'pull',  url:
↪'tcp://127.0.0.1:12000', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}
      ops_state_goal:  { name: 'ops_state_goal',  port_type: 'pull',  url:
↪'tcp://127.0.0.1:12010', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}
      srv_port:         { name: 'srv_port',         port_type: 'pull',  url:
↪'tcp://127.0.0.1:12001', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}
      query_port:       { name: 'query_port',       port_type: 'rep',   url:
↪'tcp://127.0.0.1:12002', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}

   outputs:
      ops_state_value: { name: 'ops_state_value', port_type: 'push',  url:
↪'tcp://127.0.0.1:12011', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}
      heartbeat_out:    { name: 'heartbeat_out',    port_type: 'push',  url:
↪'tcp://127.0.0.1:12012', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}
```

```
      pub_port:         { name: 'pub_port',         port_type: 'pub',    url:
↪'tcp://127.0.0.1:12003', blocking_mode: 'async', max_rate: 10, nom_rate: 1␣
↪}
```

`--database=<name>`

The name of the database to be used by the Server

`--collection=<name>`

The name of the collection to be used by the Server.

Note: Different databases or collections can be used for testing and development purposes. For the purpose of software testing with the SDK, the services have predefined databases and collections.

`--help`

Outputs the description and options of the application

`--logging=<level>`

The following table describes the different values available for the `logging` option

<level>	Abbr	Description
fatal	FTL	fatal - errors which make the application unusable
error	ERR	error - errors that preclude to achive an specific request
warning	WRN	warning - problems that may caused that the result achieved may not be the expected
info	INF	info - information about the general execution of the application
debug	DBG	debug - information to provide an understanding of the internal of the application
trace	TRC	trace - information that may server to identify a potential problem
metric	MET	metric - information to record performance metrics relative to the execution of the application

`--name=<name>`

application name

`--scope=<local|global>`

This option defines where the application should search for the *Server* and *Container* configuration file. When set to `local` the file will be search in $GMT_LOCAL/etc/conf/<service_dir>/ When set to `global` the file will be search in $GMT_GLOBAL/etc/conf/<service_dir>/

`--conf_file=<name>`

This option defines the name of the *Application* configuration file.

`--ctnr_conf_file=<name>`

This option defines the name of the application *Container* configuration file.

Note: The options `--scope`, `--conf_file`, `--ctnr_conf_file`, `--srv_conf_file` are intended for the testing of the core services and their deployment in different locations. It should not be necessary to modify them for normal operation of the SDK.

2.2 Service Client Application

Description

The <service>_client command line application starts the service client. For the purposes of developing and testing software components with the SDK it is recommended to start the service client with the default options or with the logging set to debug in case of troubleshooting.

```
λ tele_client

# -> will start the client with default options (recommended)

λ tele_client --logging debug

# -> will start the server with default options and will output debug log messages␣
↪(recommended for troubleshooting)
```

The following section shows the execution of the help command and the available options.

```
λ alarm_client --help

Usage: alarm_client_app [options]

If called without options alarm_client_app will use default values for them

Options:

--save                      saves service stream to file
--query=<query_spec>        querys the service database using the mongodb query syntax
                          e.g.: srv_client --query '{source: "component_name", level:
↪"error"}'
--topic=<topic_name>        filters service data frames matching the topic
--client_conf_file=<name> name of the client configuration file
--help                      prints help
--logging=<level>           logging level
--name=<name>               application name
--scope=<local|global>      scope to load the configuration file
--conf_file=<name>          name of the application configuration file
--ctnr_conf_file=<name>     name of the container configuration file
```

Options

--save

Saves service stream to file. The name of the files conforms to the following syntax: <service_app>_<pid>.csv where pid is the process identifier during the execution of the application. The file is saved in the directory where the application was invoked.

```
λ alarm_client --save

# -> will create the file alarm_client_app_2457.csv
```

Note: The service stream is the data stream that is published by the service server. If a topic is specified in the command line, only the messages matching the topic will be saved to the file. The service client will still output it's own logging events to the standard output.

--query=<query_spec>

Sends a query to the service server and returns the result. The query must be writen using the mongodb query syntax [https://docs.mongodb.com/manual/tutorial/query-documents/]

```
λ log_client --query '{source: "component_name", level: "error"}'

# -> will return the log events of the component "component_name" which␣
↪level is "error"
```

Note: The query capability of this command line option are very limited and intented for basic testing purposes. A more advanced interface will be provided in the OCS UI.

`--topic=<topic_name>`

filters service data frames matching the topic

`--client_conf_file=<name>`

name of the container configuration file

`--help`

prints the description and options of the application

`--logging=<level>`

See the description of the logging option in the server application above.

`--name=<name>`

application name

`--scope=<local|global>`

See the description of the `--scope` option in the server application above.

`--conf_file=<name>`

See the description of the `--conf_file` option in the server application above.

`--ctnr_conf_file=<name>`

See the description of the `--ctnr_conf_file` option in the server application above.

Note: The options `--scope`, `--conf_file`, `--ctnr_conf_file`, `--client_conf_file` are intended for the testing of the core services and their deployment in different locations. It should not be necessary to modify them for normal operation of the SDK.

2.3 Service Adapter

The interface with each service is provided by the specific implementation of the Core Framework: cpp, python, nodejs. Each framework implementation user guide describes the corresponding Service Adapter API. The following diagram shows an example of how the different parts of the Logging and Telemetry services interact with the rest of the system.

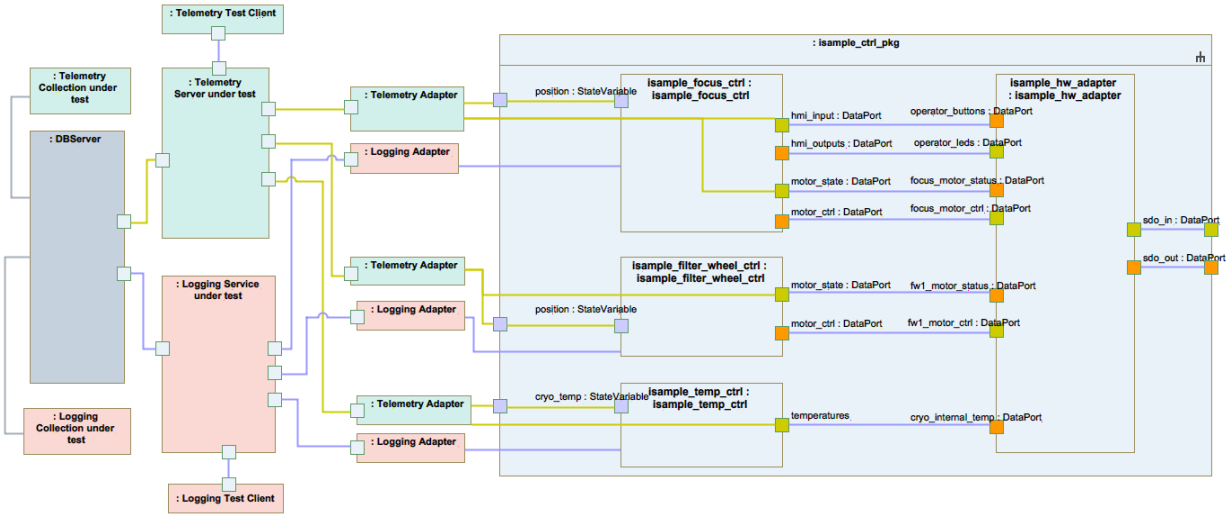

Fig. 2.1: Service operation overview

2.4 Service Database

The Services database is implemented in mongodb. The installation of mongodb is described in the SDK installation guide. Once the database is installed, the database daemon will be started automatically during the startup process of the hosting machine. The Service Servers take care of initializing the service database and collections. In addition to the installation of the mongodb daemons, the SDK also includes instructions for the installaction of the **mongo** shell, which permits more sophisticated interaction with the database. The user guide of the **mongo** shell utility is available in [https://docs.mongodb.com/manual/reference/mongo-shell/].