

---

# **grs documentation**

***Release 1.13.0***

**Software and Controls Group**

**Mar 20, 2023**

# Contents

|          |                                  |          |
|----------|----------------------------------|----------|
| <b>1</b> | <b>Introduction</b>              | <b>1</b> |
| <b>2</b> | <b>The grs command line tool</b> | <b>2</b> |
| 2.1      | grs send . . . . .               | 3        |
| 2.2      | grs listen . . . . .             | 4        |
| 2.3      | grs compile . . . . .            | 5        |
| 2.4      | grs info . . . . .               | 5        |
| 2.5      | grs get . . . . .                | 7        |
| 2.6      | grs set . . . . .                | 9        |
| 2.7      | grs inspect . . . . .            | 9        |
| 2.8      | grs db . . . . .                 | 11       |
| 2.9      | grs options . . . . .            | 12       |

## INTRODUCTION

This document describes the use of the *grs* (GMT Runtime System) tool.

## THE GRS COMMAND LINE TOOL

The `grs` utility allows interacting and communicating with remote component instances. The utility has several commands that are explained in the following sections. The `send` and `listen` commands are mostly used for debugging purposes, while the `get`, `set` and `inspect` commands are intended to interact with a remote component during operation. The `compile` command processes the configuration file of a given component so it's ready to be consumed during the start up of a component. This removes the need for sending the configuration of a component to the configuration port after starting the component.

The `grs` common options are described at the end of this document. The `grs help` command provides a brief description of the utility:

```
$ grs --help
grs [command]

Commands:
grs send [instance]          Send a message to the port of the
↳instance[aliases: push]
grs listen [instance]       Listen to messages from the port of the
↳instance
grs compile [module]        Compiles the configuration of a module or
↳compile the
                             input file passed as argument
grs info <instance>         displays configuration information for a
↳given instance
grs get <instance>          Obtains the value of a feature of a remote
↳Component
                             instance
grs set <instance>          Sets the value of a feature in a remote
↳Component
                             instance
grs db <operation> <instance> Database operations when the instance
↳implements a
                             database server
grs inspect <instance>      scans the values of all the features of a
↳remote
                             Component instance

Options:
--version    Show version number
↳ [boolean]
--name       name of the configurable entity [string] [default:
↳"grs_7074"]
--scope      scope to load the configuration file [string]
↳[default: ""]
--config     name of the component configuration to be applied
```

```

                                [string]_
↪[default: "default"]
--auto_conf  load the configuration from file if it exists      [boolean]_
↪[default: true]
--logging    logging level                                     [string]_
↪[default: "info"]
--help      Show help                                         _
↪ [boolean]

Examples:
grs send -i x_ctrl -p position_goal -m 1
grs send -i x_ctrl -p position_goal -m 1 2.4 3 -d 100
grs listen --url 'tcp://127.0.0.1:14103'
grs listen tele_server --port 'pub_port'
grs compile <instance_name>
grs compile --input <absolute_file_path> --output <file_path>
grs get x_ctrl -f state_vars
grs get x_ctrl -f state_vars/position
grs get x_ctrl -f state_vars/position/goal
grs set x_ctrl -f state_vars/position/goal -v 1.5
grs db query alarm_srv -e '{src: "alarm_srv"}'
grs db query alarm_srv -e '{src: {$regex: /alarm_srv/}}' --limit 10
grs db insert alarm_srv -r '{src: "test_client"}'
grs db update alarm_srv -e '{src: {$regex: /alarm_srv/}}' -u '{ $set:
↪{counter: 1} }'
grs db delete alarm_srv -e '{state: "ON" }'
grs inspect alarm_server
grs inspect alarm_server -f properties
grs inspect alarm_server -f state_vars -s goal

For more information, find our manual at https://gmto.github.io/gmt\_docs/

```

## 2.1 grs send

### Description

The send command sends messages to a running component instance. The receiver endpoint can be specified in two ways:

- Using the instance, the port and the configuration (this last one is optional).
- Using the url of the receiver port.

The help options displays an overview of the send command:

```

$ grs send --help

grs send [instance]

Send a message to the port of the instance

Options:
--instance, -i  name of the instance                                _
↪ [string]
--delay, -d     Delay between messages                             [number]_
↪[default: 1]
--port, -p     name of the port                                     _
↪ [string]

```

```

--url, -u      url of the port
↪ [string]
--msg, -m      message(s) to be send
↪ [array]
--file, -f     Name of the file to send
↪ [string]
--repeats, -r  Number of times the message is send [number]
↪ [default: 1]
--conf, -c     name of the configuration [string]
↪ [default: "default"]

```

### Options

**--instance, -i** the name of the receiver component instance.

**--delay, -d** the delay between messages in milliseconds if case more than one message is sent (see option **--repeats**)

**--port, -p** the name of the receiver component port

**--url, -u** the url of the receiver port

**--msg, -m** the message(s) to be send. The messages are separated by an space.

```

$ grs send -i test_cmp -p position_goal -m 1 2 3 -d 100
# will send 3 messages (1 -> a), (2 -> b), (3 -> c) with a delay of 100 ms
↪ between them

```

The following data types are supported:

- string: 'message content'
- array: '[1, 2, 3, [4, 5]]'
- struct: '{one: 1, two: "two", three: [1, 2, 3]}'
- number: 10

**--file, -f** name of the file to be send

**--repeats, -r** Number of times that the message(s) is send.

**--conf, -c** the name of the configuration of the receiver component

### Examples

## 2.2 grs listen

### Description

The `listen` command listens to messages from a running component instance Like in the `send` command is possible to address a remote endpoint by giving the instance, port and configuration or by using directly the url of the remote port

```

$ grs listen --help

grs listen [instance]

```

```

Listen to messages from the port of the instance

Options:
--instance, -i  name of the instance
↪ [string]
--port, -p      name of the port           [string] [default:
↪ "sd_rep_out"]
--url, -u       url of the port           [string]
↪ [string]
--conf, -c      name of the configuration   [string]
↪ [default: "default"]

```

### Options

**--instance, -i** the name of the remote component instance.

**--port, -p** the name of the remote component port

**--url, -u** the url of the remote port

**--conf, -c** the name of the configuration of the remote component

## 2.3 grs compile

### Description

The `compile` commands process the configuration file of a given component so it's ready to be consumed during the start up of a component. This removes the need for sending the configuration of a component to the configuration port after starting the component. The compiled configuration is saved in the same location as the configuration file (e.g: `$GMT_LOCAL/etc/conf/<subsystem>`) with the extension `.cfg`

```

$ grs compile --help

grs compile [instance]

Compile the configuration of an instance. Compiled files are saved in $GMT_
↪LOCAL/etc/conf/subsystem/

Options:
--instance, -i  name of the instance
↪ [string]
--conf, -c      name of the configuration   [string]
↪ [default: "default"]

```

### Options

**--instance, -i** the name of the component instance.

**--conf, -c** the name of the configuration of the component instance

## 2.4 grs info

### Description

The `info` command displays the configuration information of a component instance, showing what features are defined. This information can be used in the `get`, `set` commands.

```
$ grs info --help

grs info [instance]

displays configuration information for a given instance

Options:
--instance, -i  name of the instance
→ [string]
--conf, -c      name of the configuration           [string]
→ [default: "default"]
```

## Options

**--instance, -i** the name of the remote component instance.

**--conf, -c** the name of the configuration of the remote component

## Examples

```
$ grs info -i x_ctrl

Configuration: default
{ properties:
{ host:
  { name: 'host',
    default_value: '127.0.0.1',
    type: 'string',
    desc: '' },
  port:
  { name: 'port', default_value: 12200, type: 'integer', desc: '' },
  scan_rate:
  { name: 'scan_rate', default_value: 1, type: 'integer', desc: '' } },
state_vars:
{ position:
  { name: 'position',
    default_value: 1.1,
    min: -100,
    max: 100,
    type: 'float',
    blocking_mode: 'sync',
    desc: 'Axis position' } },
inputs:
{ encoder:
  { name: 'encoder',
    default_value: 1.1,
    min: -100,
    max: 100,
    type: 'float',
    blocking_mode: 'sync',
    desc: 'Axis encoder input' } },
outputs:
{ motor:
  { name: 'motor',
    default_value: 1.1,
    min: -100,
    max: 100,
    type: 'float',
    blocking_mode: 'sync',
```



```

        desc: 'Axis motor demand' } },
faults:
{ motor_over_heat:
  { name: 'motor_over_heat',
    default_value: 'NOT_ACTIVE',
    kind: 'primary',
    level: 'CRITICAL',
    detection_latency: 1,
    type: 'string',
    parent: 'axis_fault',
    desc: 'Motor Overheat' },
encoder_fault:
  { name: 'encoder_fault',
    default_value: 'NOT_ACTIVE',
    kind: 'primary',
    level: 'CRITICAL',
    detection_latency: 1,
    type: 'string',
    parent: 'axis_fault',
    desc: 'Encoder not responding' },
motor_fault:
  { name: 'motor_fault',
    default_value: 'NOT_ACTIVE',
    kind: 'primary',
    level: 'CRITICAL',
    detection_latency: 1,
    type: 'string',
    parent: 'axis_fault',
    desc: 'Motor not responding' },
axis_fault:
  { name: 'axis_fault',
    default_value: 'NOT_ACTIVE',
    kind: 'or',
    level: 'CRITICAL',
    detection_latency: 1,
    type: 'string',
    parent: null,
    desc: 'Axis fault' } },
alarms:
{ axis_malfunction_alarm:
  { name: 'axis_malfunction_alarm',
    default_value: 'NORM',
    type: 'string',
    desc: 'The X Axis is not operational' } },
connectors: []
Proxy ports:
{ sd_req_in: 'tcp://127.0.0.1:12204',
sd_rep_out: 'tcp://127.0.0.1:12203',
sd_req_out: 'tcp://127.0.0.1:12201',
sd_rep_in: 'tcp://127.0.0.1:12202' }

```

## 2.5 grs get

### Description

The `get` command retrieves information about the state of a component. If not feature is specified it will display all

the features of the component.

```
$ grs get --help

grs get [instance]

displays the value of a feature of a remote Component instance

Options:
--instance, -i  name of the instance
↪ [string]
--conf, -c      name of the configuration          [string]
↪ [default: "default"]
--feature, -f   name of the feature
↪ [string]
--slice, -s     name of the attribute to slice the collection
↪ [string]
```

### Options

**--instance, -i** the name of the remote component instance.

**--conf, -c** the name of the configuration of the remote component

**--feature, -f** the name of the feature to be get from the remote component. The feature is specified by defining its path: <feature\_set>/<feature\_name>/<feature\_attribute>, where

- feature\_set: the category of the component feature, eg.: state\_vars, inputs
- feature\_name: the name of the feature, e.g: op\_state
- feature\_attribute: the name of the attribute of the feature e.g: value, goal, desc

**--slice, -s** the name of the slice

### Examples

```
$ grs get -i x_ctrl

# will display all the features of the x_ctrl component

$ grs get -i x_ctrl -f state_vars

# will display all the values of all the state variables of x_ctrl

$ grs get -i x_ctrl -f state_vars -s desc

# will display all the descriptions of all the state variables of x_ctrl

$ grs get -i x_ctrl -f state_vars/position

# will display all the attributes of the x_ctrl position state variable

$ grs get -i x_ctrl -f state_vars/position/goal

# will display the goal of the x_ctrl position state variable
```

## 2.6 grs set

### Description

The `set` command sets the value of a feature in a remote component

```
$ grs set --help

grs set [instance]

Sets the value of a feature in a remote Component instance

Options:
--instance, -i  name of the instance
↪ [string]
--conf, -c      name of the configuration           [string]
↪ [default: "default"]
--feature, -f   name of the feature
↪ [string]
--value, -v     New value
↪ [array]
```

### Options

**--instance, -i** the name of the remote component instance.

**--conf, -c** the name of the configuration of the remote component

**--feature, -f** the name of the feature to be set in the remote component

**--value, -v** the new value to be set in the remote component feature

```
$ grs set -i x_ctrl -f state_vars/position/goal -v 1.5

# will set the goal of the remote component position state variable to 1.5
```

The following data types are supported:

- string: 'value content'
- array: '[1, 2, 3, [4, 5]]'
- struct: '{one: 1, two: "two", three: [1, 2, 3]}'
- number: 10

## 2.7 grs inspect

### Description

The `inspect` continuously retrieve the values of all the features of a remote component instance

```
$ grs inspect --help

grs inspect [instance]

retrieves the values of all the features of a remote Component instance

Options:
```

```

--instance, -i  name of the instance
↪ [string]
--feature, -f  name of the feature
↪ [string]
--rate, -r      Scan rate, 0 = 1 shot                [number]
↪ [default: 10]
--conf, -c      name of the configuration            [string]
↪ [default: "default"]

```

## Options

**--instance, -i** the name of the remote component instance.

**--feature, -f** the name of the feature to retrieve from the remote component. If this parameter is passed `grs inspect` will update only the value of the specified feature

**--rate, -r** the rate at which `grs inspect` will retrieve and display the remote information

**--conf, -c** the name of the configuration of the remote component

The following example shows the output of the `inspect` command for a component implemented in nodeJS:

```

$ grs inspect cartesian_ctrl

Timestamp: Mon Jul 01 2019 17:37:21 GMT-0700 (Pacific Daylight Time) hb
↪ #: [0]
{ properties:
  { name: 'cartesian_ctrl',
    scope: '',
    config: 'default',
    auto_conf: true,
    uri: '',
    host: '127.0.0.1',
    port: 12500,
    scan_rate: 1,
    auto_start: false,
    auto_init: false,
    auto_halt: true,
    auto_shutdown: true },
  state_vars: { op_state: 'RUNNING', position: 0.01 },
  inputs:
  { x_position_in: 18.499999999999996,
    y_position_in: 1.1,
    z_position_in: 18.499999999999996 },
  outputs:
  { x_position_out: 17.699999999999998,
    y_position_out: 17.699999999999998,
    z_position_out: 17.699999999999998 },
  faults:
  { x_axis_fault: 'NOT_ACTIVE',
    y_axis_fault: 'NOT_ACTIVE',
    z_axis_fault: 'NOT_ACTIVE',
    stage_fault: 'NOT_ACTIVE' },
  alarms: {},
  connectors: { },
  proxies: { x_ctrl: null, y_ctrl: null, z_ctrl: null } }
Enter <CTRL-C> to exit

```

## 2.8 grs db

### Description

The db allows to interact with a Component that implements the ServerProxy API.

```
$ grs db --help
grs db <operation> <instance>

Database operations when the instance implements a database server

Options:
--instance      name of the instance
↪ [string]
--conf, -c      name of the configuration [string]
↪ [default: "default"]
--operation     name of the database operation
                [string] [choices: "query", "insert", "update",
↪ "delete"]
--output, -o    output format of the database result operation
                [string] [choices: "raw", "payload", "console", "json", "json_raw"]
↪ [default: "console"]
--expr, -e      the expresion of the query in MongoDB format [string]
↪ [default: ""]
--record, -r    the record to insert in the database [string]
↪ [default: ""]
--update, -u    update to apply to the query record [string]
↪ [default: ""]
--limit, -l     the maximum number of records to return from the query
                [number]
↪ [default: 40]
--timeout, -t   timeout for the get command [number]
↪ [default: 400]
```

### Options

**--operation** the name of the database operation. The choices are:

- query: Sends a query to the service server and returns the result. The query must be written using the mongodb query syntax [<https://docs.mongodb.com/manual/tutorial/query-documents/>]
- insert: Inserts a new record in the database
- update: Updates the record(s) defined by the `--expr` option with the update expresion. The `--update` and `--expr` options follow the MongoDB syntax
- delete: Deletes the record(s) defined by the `--expr` option.

**--instance** the name of the remote component instance.

**--conf, -c** the name of the configuration of the remote component

**--output, -o** the ouput format of the database result operation. The possible choices are:

- raw: Writes to the standard output the query result as generated by the MongoDB database. It includes additional information related to the database organization
- payload: Writes to the standard output the data part of the query result.
- console: Similar with payload but with console formatting.
- json: Creates a file in the current directory with the data part of the query result.

- **json\_raw**: Creates a file in the current directory with the query result as generated by the MongoDB database.

**--expr, -e** the query expression used for the query, update and delete operations. The query expression follows the MongoDB syntax

**--record, -r** the record to be inserted in the database

**--update, -u** the update expression for the update operation. The update expression follows the MongoDB syntax

**--limit, -l** the maximum number of records returned by the database query operation

The following examples show different db commands

```
$ grs db query alarm_srv -e '{src: "alarm_srv"}'
$ grs db query alarm_srv -e '{src: {$regex: /alarm_srv/}}' --limit 10
$ grs db insert alarm_srv -r '{src: "test_client"}'
$ grs db update alarm_srv -e '{src: {$regex: /alarm_srv/}}' -u '{ $set:
↪{counter: 1} }'
$ grs db delete alarm_srv -e '{state: "ON" }'
```

## 2.9 grs options

The following options can be used in combination with the previous commands

**-V, --version** The grs command will print the version number

**-l, --logging <level>** Activates <level> logging. 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 achieve an specific request                                     |
| warn-<br>ing | 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 |

**auto\_conf** Loads and applies the grs application configuration file if it exists.

**config** Name of the grs application configuration file

**scope** Scope used to load the grs configuration file

**-h, --help** output usage information