Central Node Mid Documentation

Release 1.0

NCRA India

May 31, 2022

GETTING STARTED

This project is developing the CentralNode (Mid and Low) component of the Telescope Monitoring and Control (TMC) prototype, for the Square Kilometre Array.

CHAPTER

ONE

GETTING STARTED

This page contains instructions for software developers who want to get started with usage and development of the CentralNode.

1.1 Background

Detailed information on how the SKA Software development community works is available at the SKA software developer portal. There you will find guidelines, policies, standards and a range of other documentation.

1.2 Set up your development environment

This project is structured to use k8s for development and testing so that the build environment, test environment and test results are all completely reproducible and are independent of host environment. It uses `make` to provide a consistent UI (run `make help` for targets documentation).

1.2.1 Install minikube

You will need to install *minikube* or equivalent k8s installation in order to set up your test environment. You can follow the instruction here:

```
git clone git@gitlab.com:ska-telescope/sdi/deploy-minikube.git
cd deploy-minikube
make all
eval $(minikube docker-env)
```

Please note that the command `eval $(minikube docker-env)` will point your local docker client at the docker-in-docker for minikube. Use this only for building the docker image and another shell for other work.

1.2.2 How to Use

Clone this repo:

```
git clone https://gitlab.com/ska-telescope/ska-tmc-centralnode.git
cd ska-tmc-centralnode
```

Install dependencies

```
apt update
apt install -y curl git build-essential libboost-python-dev libtango-dev
curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py |␣
↪python3 -
source $HOME/.poetry/env
```

Please note that:

- the *libtango-dev* will install an old version of the TANGO-controls framework (9.2.5);
- the best way to get the framework is compiling it (instructions can be found here);
- the above script has been tested with Ubuntu 20.04.

During this step, `libtango-dev` instalation can ask for the Tango Server IP:PORT. Just accept the default proposed value.

Install python requirements for linting and unit testing:

```
$ poetry install
```

Activate the poetry environment:

```
$ source $(poetry env info --path)/bin/activate
```

Alternate way to install and activate poetry

Follow the steps till installation of dependencies. then,

$ virtualenv cn_venv $ source cn_venv/bin/activate $ make requirements

Run python-test:

```
$ make python-test
PyTango 9.3.3 (9, 3, 3)
PyTango compiled with:
    Python : 3.8.5
    Numpy  : 0.0.0 ## output generated from a WSL windows machine
    Tango  : 9.2.5
    Boost  : 1.71.0

PyTango runtime is:
    Python : 3.8.5
    Numpy  : None
    Tango  : 9.2.5

PyTango running on:
uname_result(system='Linux', node='LAPTOP-5LBGJH83', release='4.19.128-microsoft-standard
↪', version='#1 SMP Tue Jun 23 12:58:10 UTC 2020', machine='x86_64', processor='x86_64')
```

(continues on next page)

(continued from previous page)

```
============================= test session starts ==============================
platform linux -- Python 3.8.5, pytest-5.4.3, py-1.10.0, pluggy-0.13.1 -- /home/
[....]

---------------------------------- JSON report ----------------------------------
JSON report written to: build/reports/report.json (165946 bytes)

----------- coverage: platform linux, python 3.8.5-final-0 -----------
Coverage HTML written to dir build/htmlcov
Coverage XML written to file build/reports/code-coverage.xml

======================== 48 passed, 5 deselected in 42.42s =======================
```

Formatting the code:

```
$ make python-format
[...]
--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

Python linting:

```
$ make python-lint
[...]
--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

Helm Charts linting:

```
$ make helm-lint
[...]
10 chart(s) linted, 0 chart(s) failed
```

Build the container image for the project:

```
$ make oci-build
[...]
[+] Building 111.7s (14/14) FINISHED
[...]
```

Install the umbrella chart:

```
$ make k8s-install-chart
[...]
NAME: test
LAST DEPLOYED: Fri Nov  5 10:35:18 2021
NAMESPACE: ska-tmc-centralnode
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Test the deployment with (the result of the tests are stored into the folder `charts/build`):

```
$ make k8s-wait && make k8s-test
k8s-test: start test runner: test-runner-test -n ska-tmc-centralnode
k8s-test: sending test folder: tar -cz src/ tests/
( cd /home/ubuntu/ska-tmc-centralnode; tar -cz src/ tests/ \
| kubectl run test-runner-test -n ska-tmc-centralnode --restart=Never --pod-running-
↪timeout=360s  --image-pull-policy=IfNotPresent --image=artefact.skao.int/ska-tmc-
↪centralnode:0.3.4-dirty --env=INGRESS_HOST=  -iq -- /bin/bash -o pipefail -c " mkfifo␣
↪results-pipe && tar zx --warning=all && cd tests && ( if [[ -f requirements.txt ]];␣
↪then echo 'k8s-test: installing requirements.txt'; pip install -qUr requirements.txt;␣
↪fi ) && export PYTHONPATH=:/app/src:/app/ska_tmc_centralnode/ && mkdir -p build && (␣
↪cd .. && PYTHONPATH=.:./src TANGO_HOST=tango-databaseds:10000  pytest -m 'SKA_mid and␣
↪(post_deployment or acceptance)'  --true-context tests ./tests | tee pytest.stdout;  );
↪ echo \$? > build/status; pip list > build/pip_list.txt; echo \"k8s_test_command: test␣
↪command exit is: \$(cat build/status)\"; tar zcf ../results-pipe build;" 2>&1 \
| grep -vE "^(1\||-+ live log)" --line-buffered &); \
sleep 1; \
echo "k8s-test: waiting for test runner to boot up: test-runner-test -n ska-tmc-
↪centralnode"; \
( \
kubectl wait pod test-runner-test -n ska-tmc-centralnode --for=condition=ready --
↪timeout=360s ; \
wait_status=$?; \
if ! [[ $wait_status -eq 0 ]]; then echo "Wait for Pod test-runner-test -n ska-tmc-
↪centralnode failed - aborting"; exit 1; fi; \
) && \
        echo "k8s-test: test-runner-test -n ska-tmc-centralnode is up, now waiting for␣
↪tests to complete" && \
        (kubectl exec test-runner-test -n ska-tmc-centralnode -- cat results-pipe | tar -
↪-directory=/home/ubuntu/ska-tmc-centralnode -xz); \
\
cd /home/ubuntu/ska-tmc-centralnode/; \
(kubectl get all,job,pv,pvc,ingress,cm -n ska-tmc-centralnode -o yaml > build/k8s_
↪manifest.txt); \
echo "k8s-test: test run complete, processing files"; \
kubectl --namespace ska-tmc-centralnode delete --ignore-not-found pod test-runner-test --
↪wait=false
k8s-test: waiting for test runner to boot up: test-runner-test -n ska-tmc-centralnode
pod/test-runner-test condition met
k8s-test: test-runner-test -n ska-tmc-centralnode is up, now waiting for tests to␣
↪complete
k8s-test: installing requirements.txt
[...]
====================== 19 passed, 178 deselected in 18.38s ======================
```

It is possible to install and test the centralnode both in Mid and Low by passing the variable TELESCOPE (`SKA-mid` or `SKA-low`):

```
$ make k8s-install-chart TELESCOPE=SKA-low
$ make k8s-wait TELESCOPE=SKA-low
$ make k8s-test TELESCOPE=SKA-low
```

Uninstall the chart:

```
$ make k8s-uninstall-chart
release "test" uninstalled
```

1.2.3 Makefile targets

This project contains a Makefile which acts as a UI for building container images, testing images, and for launching interactive developer environments. For the documentation of the Makefile run `make help`.

CHAPTER

TWO

CENTRALNODE CODE QUALITY GUIDELINES

2.1 Code formatting / style

2.1.1 Black

CentralNode uses the `black` code formatter to format its code. Formatting can be checked using the command `make python-format`.

The CI pipeline does check that if code has been formatted using black or not.

2.1.2 Linting

CentralNode uses below libraries/utilities for linting. Linting can be checked using command `make python-lint`.

- **`isort` - It provides a command line utility, Python library and** plugins for various editors to quickly sort all your imports.
- `black` - It is used to check if the code has been blacked.
- **`flake8` - It is used to check code base against coding style (PEP8),** programming errors (like "library imported but unused" and "Undefined name"),etc.
- **`pylint` - It is looks for programming errors, helps enforcing a coding standard,** sniffs for code smells and offers simple refactoring suggestions.

2.2 Test coverage

CentralNode uses pytest to test its code, with the pytest-cov plugin for measuring coverage. .

CHAPTER

THREE

API

3.1 ska_tmc_centralnode package

3.1.1 Subpackages

ska_tmc_centralnode.commands package

Submodules

ska_tmc_centralnode.commands.abstract_command module

class ska_tmc_centralnode.commands.abstract_command.**AbstractAssignReleaseResources**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

check_allowed_low()

Checks whether this command is allowed to be run in current device state

Returns True if this command is allowed to be run in current device state

Return type boolean

Raises DevFailed if this command is not allowed to be run in current device state

check_allowed_mid()

Checks whether this command is allowed to be run in current device state

Returns True if this command is allowed to be run in current device state

Return type boolean

Raises DevFailed if this command is not allowed to be run in current device state

init_adapters_low()

init_adapters_mid()

class ska_tmc_centralnode.commands.abstract_command.**AbstractTelescopeOnOff**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

check_allowed_low()

Checks whether this command is allowed It checks that the device is in a state to perform this command and that all the component needed for the operation are not unresponsive

Returns True if this command is allowed

Return type boolean

check_allowed_mid()
Checks whether this command is allowed It checks that the device is in a state to perform this command and that all the component needed for the operation are not unresponsive

Returns True if this command is allowed

Return type boolean

init_adapters_low()

init_adapters_mid()

class ska_tmc_centralnode.commands.abstract_command.**CentralNodeCommand**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

check_allowed()

do(*argin=None*)

init_adapters()

invoke_command(*adapters: list*, *command_caller*, *err_msg: str*)

send_command(*adapters*, *description*, *command*, *argin=None*)

ska_tmc_centralnode.commands.assign_resources_command module

AssignResources class for CentralNode.

class ska_tmc_centralnode.commands.assign_resources_command.**AssignResources**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

A class for CentralNode's AssignResources() command.

Assigns resources to given subarray. It accepts the subarray id, receptor id list and SDP block in JSON string format. Upon successful execution, the 'receptor_ids' attribute of the given subarray is populated with the given receptors.Also checking for duplicate allocation of resources is done. If already allocated it will throw error message regarding the prior existence of resource.

create_mccs_cmd_data(*json_argument*)
Remove 'sdp' and 'mccs' key from input JSON argument and forward the updated JSON to mccs master leaf node.

Parameters **json_argument** – The string in JSON format.

Returns The string in JSON format.

create_subarray_cmd_data(*json_argument*)
Remove 'subarray id', 'sdp' from json argument and forward the updated JSON to Subarray node.

Parameters **json_argument** – The string in JSON format.

Returns The string in JSON format.

do_low(*argin=None*)
Method to invoke AssignResources command on Subarray.

Parameters **argin** – The string in JSON format. The JSON contains following values: interface:

DevString. Mandatory. Version of schema to allocate assign resources.

subarray_id: DevShort. Mandatory. Sub-Array to allocate resources to

mccs:

subarray_beam_ids: DevArray. Mandatory logical ID of beam

station_ids: DevArray. Mandatory list of stations contributing beams to the data set

channel_blocks: DevArray. Mandatory list of channels used

Example

{"interface":"https://schema.skao.int/ska-low-tmc-assignresources/2.0","transaction_id":"txn-....-00001","subarray_id":1,"mccs":{"subarray_beam_ids":[1],"station_ids":[[1,2]],"channel_blocks":[3]},"sdp":{}}

Note: Enter input without spaces as: {"interface":"https://schema.skao.int/ska-low-tmc-assignresources/2.0", "transaction_id":"txn-....-00001","subarray_id":1,"mccs":{"subarray_beam_ids":[1], "station_ids":[[1,2]], "channel_blocks":[3]},"sdp":{}} :returns: None

Raises

- **KeyError if input argument json string contains invalid key** –
- **ValueError if input argument json string contains invalid value** –
- **AssertionError if Mccs On command is not completed.** –

do_mid(*argin=None*)

Method to invoke AssignResources command on Subarray.

Parameters **argin** – The string in JSON format. The JSON contains following values:

subarray_id: DevShort. Mandatory.

dish: Mandatory JSON object consisting of

receptor_ids: DevVarStringArray The individual string should contain dish numbers in string format with preceding zeroes upto 3 digits. E.g. 0001, 0002.

sdp: Mandatory JSON object consisting of

eb_id: DevString The SBI id.

max_length: DevDouble Maximum length of the SBI in seconds.

scan_types: array of the blocks each consisting following parameters scan_type_id:

DevString The scan id.

coordinate_system: DevString

ra: DevString

Dec: DevString

processing_blocks: array of the blocks each consisting following parameters

eb_id: DevString The Processing Block id.

workflow:

kind: DevString

name: DevString

version: DevString

parameters: {}

Example

{“interface”:”https://schema.skao.int/ska-tmc-assignresources/2.0”, “transaction_id”:”txn-....-00001”,”subarray_id”:1,”dish”: {“receptor_ids”:[“0001”,”0002”]},”sdp”:{“interface”: “https://schema.skao.int/ska-sdp-assignres/0.3”,”eb_id”: “eb-mvp01-20200325-00001”,”max_length”:100.0,”scan_types”: [{“scan_type_id”:”science_A”,”reference_frame”:”ICRS”,”ra”:”02:42:40.771” ,”dec”:”-00:00:47.84”,”channels”:[{“count”:744,”start”:0,”stride”:2,”freq_min” :0.35e9,”freq_max”:0.368e9,”link_map”:[[0,0],[200,1],[744,2],[944,3]]}, {“count”:744,”start”:2000,”stride”:1,”freq_min”:0.36e9,”freq_max”:0.368e9, “link_map”:[[2000,4],[2200,5]]}]},{“scan_type_id”:”calibration_B”,”reference_frame” :”ICRS”,”ra”:”12:29:06.699”,”dec”:”02:03:08.598”,”channels”:[{“count”:744, “start”:0,”stride”:2,”freq_min”:0.35e9,”freq_max”:0.368e9,”link_map”: [[0,0],[200,1],[744,2],[944,3]]},{“count”:744,”start “freq_min”:0.36e9,”freq_max”:0.368e9,”link_map”:[[2000,4], [2200,5]]}]}],”processing_blocks”:[{“pb_id”:”pb-mvp01-20200325-00001” ,”workflow”:{“kind”:”realtime”,”name”:”vis_receive”,”version”:”0.1.0”}, “parameters”:{}},{“pb_id”:”pb-mvp01-20200325-00002”,”workflow”:{“kind”:”realtime”, “name”:”test_realtime”,”version”:”0.1.0”},”parameters”:{}},{“pb_id”:”pb-mvp01-20200325-00003”, “workflow”:{“kind”:”batch”,”name”:”ical”,”version”:”0.1.0”},”parameters”:{}, “dependencies”:[{“pb_id”:”pb-mvp01-20200325-00001”,”kind”:[“visibilities”]}]}, {“pb_id”:”pb-mvp01-20200325-00004”,”workflow”:{“kind”:”batch”,”name”:”dpreb”,”version”: “0.1.0”},”parameters”:{},”dependencies”:[{“pb_id”:”pb-mvp01-20200325-00003”,”kind”:[“calibration”]}]}]}}

Note: From Jive, enter above input string without any space.

Returns

A tuple containing a return code and a string in JSON format on successful assignment of given resources. The JSON string contains following values:

dish: Mandatory JSON object consisting of

receptor_ids_allocated: DevVarStringArray Contains ids of the receptors which are successfully allocated. Empty on unsuccessful allocation.

Example

{ “dish”: { “receptor_ids_allocated”: [“0001”] } }

Note: Enter input without spaces as:{“dish”:{“receptor_ids_allocated”:[“0001”]}}

Returns None

`get_subarray_adapter`(*subarray_id*)

`update_resource_config_file`(*json_argument*, *id*)

This method utilizes SKUID service to generate unique sb_id / eb_id and pb_id

ska_tmc_centralnode.commands.release_resources_command module

ReleaseResources class for CentralNode.

class ska_tmc_centralnode.commands.release_resources_command.**ReleaseResources**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

A class for CentralNode's ReleaseResources() command.

Release all the resources assigned to the given Subarray. It accepts the subarray id, releaseALL flag and receptorIDList in JSON string format. When the releaseALL flag is True, ReleaseAllResources command is invoked on the respective SubarrayNode. In this case, the receptorIDList tag is empty as all the resources of the Subarray are to be released. When releaseALL is False, ReleaseResources will be invoked on the SubarrayNode and the resources provided in receptorIDList tag, are to be released from the Subarray. The selective release of the resources when releaseALL Flag is False is not yet supported.

do_low(*argin*)

Method to invoke ReleaseResources command on Subarray Node.

Parameters **argin** – The string in JSON format. The JSON contains following values:

subarray_id: DevShort. Mandatory.

release_all: Boolean(True or False). Mandatory. True when all the resources to be released from Subarray.

Example: {"interface":"https://schema.skao.int/ska-low-tmc-releaseresources/2.0","transaction_id":"txn-....-00001","subarray_id":1,"release_all":true}

Note: From Jive, enter input as: {"interface":"https://schema.skao.int/ska-low-tmc-releaseresources/2.0","transaction_id":"txn-....-00001","subarray_id":1,"release_all":true} without any space.

Returns None

Raises

- **ValueError if input argument json string contains invalid value** –
- **KeyError if input argument json string contains invalid key** –
- **DevFailed if the command execution or command invocation on SubarrayNode is not successful** –

do_mid(*argin*)

Method to invoke ReleaseResources command on Subarray.

Parameters **argin** – The string in JSON format. The JSON contains following values:

subarray_id: DevShort. Mandatory.

release_all: Boolean(True or False). Mandatory. True when all the resources to be released from Subarray.

receptor_ids: DevVarStringArray. Empty when release_all tag is True.

Example:

{ "interface": "https://schema.skao.int/ska-tmc-releaseresources/2.0", "transaction_id": "txn-....-00001", "subarray_id": 1, "release_all": true, "receptor_ids": [ ]

}

Note: From Jive, enter input as: {“interface”:”https://schema.skao.int/ska-tmc-releaseresources/1.0”, “subarray_id”:1,”release_all”:true,”receptor_ids”:[]}

Returns None

release_all_resources(*adapter*)

release_resources_mccs(*arg*)

validate_input_json(*argin*)

ska_tmc_centralnode.commands.stow_antennas_command module

class ska_tmc_centralnode.commands.stow_antennas_command.**StowAntennas**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

A class for CentralNode’s StowAntennas() command.

Invokes the command SetStowMode on the specified receptors.

check_allowed()

Checks whether this command is allowed It checks that the device is in a state to perform this command and that all the component needed for the operation are not faulty

Returns True if this command is allowed

Return type boolean

do(*argin*)

Method to invoke StowAntennas command.

param argin: List of Receptors to be stowed.

init_adapters()

set_stow_mode_dishes(*adapters*)

ska_tmc_centralnode.commands.telescope_off_command module

class ska_tmc_centralnode.commands.telescope_off_command.**TelescopeOff**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

A class for CentralNode’s TelescopeOff() command. Sets the CentralNode into telescopestate to OFF.

do_low(*argin=None*)

Method to invoke telescopeoff command on Lower level devices. param:

None

Returns A tuple containing a return code and a string message indicating status.

rtype: (ResultCode, str)

do_mid(*argin=None*)

Method to invoke telescopeoff command on Lower level devices. param:

None

Returns A tuple containing a return code and a string message indicating status.

rtype: (ResultCode, str)

set_standby_fp_mode_dishes()

set_standby_lp_mode_dishes()

turn_off_csp()

turn_off_mccs_mln()

turn_off_sdp()

turn_off_subarrays()

ska_tmc_centralnode.commands.telescope_on_command module

class ska_tmc_centralnode.commands.telescope_on_command.**TelescopeOn**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

A class for CentralNode's TelescopeOn() command.

TelescopeOn command on Central node enables the telescope to perform further operations and observations. It Invokes On command on lower level devices.

do_low(*argin=None*)
Method to invoke Telescope On command on Lower level devices.

param argin: None.

do_mid(*argin=None*)
Method to invoke Telescope On command on Lower level devices.

param argin: None.

set_operate_mode_dishes()

set_standby_fp_mode_dishes()

turn_on_csp()

turn_on_mccs_master()

turn_on_sdp()

turn_on_subarrays()

ska_tmc_centralnode.commands.telescope_standby_command module

class ska_tmc_centralnode.commands.telescope_standby_command.**TelescopeStandby**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_command.ska_tmc_common.tmc_command.TMCCommand._name

A class for CentralNode's TelescopeStandby() command.

do_low(*argin=None*)
Method to invoke TelescopeStandby command on Lower level devices.

param: None

Returns none

do_mid(*argin=None*)
Method to invoke TelescopeStandby command on Lower level devices.

param: None

Returns none

set_standby_fp_mode_dishes()

set_standby_lp_mode_dishes()

turn_standby_csp()

turn_standby_mccs()

turn_standby_sdp()

turn_standby_subarrays()

Module contents

ska_tmc_centralnode.manager package

Submodules

ska_tmc_centralnode.manager.aggregators module

class ska_tmc_centralnode.manager.aggregators.**HealthStateAggregatorLow**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.aggregators.ska_tmc_common.aggregators.Aggregator._name

aggregate()

class ska_tmc_centralnode.manager.aggregators.**HealthStateAggregatorMid**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.aggregators.ska_tmc_common.aggregators.Aggregator._name

aggregate()

class ska_tmc_centralnode.manager.aggregators.**TMCOpStateAggregator**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.aggregators.ska_tmc_common.aggregators.Aggregator._name

aggregate()

class ska_tmc_centralnode.manager.aggregators.**TelescopeStateAggregatorLow**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.aggregators.ska_tmc_common.aggregators.Aggregator._name

aggregate()

class ska_tmc_centralnode.manager.aggregators.**TelescopeStateAggregatorMid**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.aggregators.ska_tmc_common.aggregators.Aggregator._name

aggregate()

ska_tmc_centralnode.manager.component_manager module

This module provided a reference implementation of a BaseComponentManager.

It is provided for explanatory purposes, and to support testing of this package.

class ska_tmc_centralnode.manager.component_manager.**CNComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: ska_tango_base.base.ska_tango_base.base.BaseComponentManager._name

A component manager for The Central Node component.

It supports:

- Monitoring its component, e.g. detect that it has been turned off or on
- Fetching the latest SCM indicator values of the components periodically and trigger the TMC and telescope state aggregation
- Receiving the change events from the component and trigger the TMC and telescope state aggregation

add_device(*dev_name*)

Add device to the monitoring loop

Parameters **dev_name** (*str*) – device name

add_dishes(*dln_prefix*, *num_dishes*)

Add dishes to the monitoring loop

Parameters

- **dln_prefix** (*str*) – prefix of the dish
- **num_dishes** (*int*) – number of dishes

add_multiple_devices(*device_list*)

Add multiple devices to the monitoring loop

Parameters **device_list** – list of device names

check_if_csp_mln_is_responsive()

check_if_dishes_are_responsive()

check_if_mccs_mln_is_responsive()

check_if_sdp_mln_is_responsive()

check_if_subarrays_are_responsive()

property checked_devices

Return the list of the checked monitored devices

Returns list of the checked monitored devices

property command_executed

property command_executor

property command_in_progress

property component

Return the managed component

Returns the managed component

Return type Component

device_failed(*device_info*, *exception*)

Set a device to failed and call the relative callback if available

Parameters

- **device_info** (*DeviceInfo*) – a device info
- **exception** – an exception

Type Exception

property devices

Return the list of the monitored devices

Returns list of the monitored devices

get_device(*dev_name*)

Return the device info our of the monitoring loop with name dev_name

Parameters **dev_name** (*str*) – name of the device

Returns a device info

Return type DeviceInfo

get_telescope_health_state()

get_telescope_state()

get_tmc_op_state()

property input_parameter

Return the input parameter

Returns input parameter

Return type *InputParameter*

is_already_assigned(*dishId*)

Check if a Dish is already assigned to a subarray

Parameters **dishId** (*str*) – id of the dish

:return True is already assigned, False otherwise

reset()

set_aggregators(*_telescope_state_aggregator*, *_health_state_aggregator*, *_tm_op_state_aggregator*)

stop()

update_device_health_state(*dev_name*, *health_state*)

Update a monitored device health state aggregate the health states available

Parameters

- **dev_name** (*str*) – name of the device
- **health_state** (*HealthState*) – health state of the device

update_device_info(*device_info*)

Update a device with correct monitoring information and call the relative callback if available

Parameters **device_info** (*DeviceInfo*) – a device info

update_device_obs_state(*dev_name*, *obs_state*)

Update a monitored device obs state, and call the relative callbacks if available

Parameters

- **dev_name** (*str*) – name of the device

- **obs_state** (*ObsState*) – obs state of the device

update_device_state(*dev_name*, *state*)

Update a monitored device state, aggregate the states available and call the relative callbacks if available

Parameters

- **dev_name** (*str*) – name of the device
- **state** (*DevState*) – state of the device

update_event_failure(*dev_name*)

update_input_parameter()

ska_tmc_centralnode.manager.monitoring_loop module

class ska_tmc_centralnode.manager.monitoring_loop.**CentralNodeMonitoringLoop**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.monitoring_loop.ska_tmc_common.monitoring_loop.MonitoringLoop._name

The MonitoringLoop class has the responsibility to monitor the sub devices managed by the central node.

It is an infinite loop which ping, get the state, the obsState, the healthState and device information of the monitored SKA devices

TBD: what about scalability? what if we have 1000 devices?

create_device_info(*devInfo*, *proxy*)

device_task(*dev_info*)

get_assignedResources_attributes(*proxy*)

Module contents

ska_tmc_centralnode.model package

Submodules

ska_tmc_centralnode.model.component module

class ska_tmc_centralnode.model.component.**CentralComponent**(**args: Any*, ***kwargs: Any*)

Bases: ska_tmc_common.tmc_component_manager.ska_tmc_common.tmc_component_manager.TmcComponent._name

A component class for Central Node

It supports:

- Maintaining a connection to its component
- Monitoring its component

property desired_telescope_state

Return desired telescope state

Returns desired telescope state

Return type DevState

property devices
Return the monitored devices.

Returns the monitored devices

Return type DeviceInfo[]

get_device(*dev_name*)
Return the monitored device info by name.

Parameters **dev_name** – name of the device

Returns the monitored device info

Return type DeviceInfo

property imaging
Return vlbi ModesAvailability

Returns vlbi ModesAvailability

Return type *ModesAvailability*

property pss
Return pss ModesAvailability

Returns pss ModesAvailability

Return type *ModesAvailability*

property pst
Return pss ModesAvailability

Returns pss ModesAvailability

Return type *ModesAvailability*

remove_device(*dev_name*)
Remove a device from the list

Parameters **dev_name** – name of the device

set_op_callbacks(*_update_device_callback=None*, *_update_telescope_state_callback=None*, *_update_telescope_health_state_callback=None*, *_update_tmc_op_state_callback=None*, *_update_imaging_callback=None*)

property telescope_health_state
Return the telescope health state

Returns the telescope health state

Return type HealthState

property telescope_state
Return the telescope state

Returns the telescope state

Return type DevState

property tmc_op_state
Return the TMC operational State

Returns the TMC operational State

Return type DevState

to_dict()

to_json()

update_device(*dev_info*)
Update (or add if missing) Device Information into the list of the component.

Parameters **dev_info** – a DeviceInfo object

update_device_exception(*dev_info*, *exception*)
Update (or add if missing) Device Information into the list of the component.

Parameters **dev_info** – a DeviceInfo object

property vlbi
Return vlbi ModesAvailability

Returns vlbi ModesAvailability

Return type *ModesAvailability*

class ska_tmc_centralnode.model.component.**MCCSDeviceInfo**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.device_info.ska_tmc_common.device_info.DeviceInfo._name

from_dev_info(*mccsdev_info*)

to_dict()

to_json()

ska_tmc_centralnode.model.component.**dev_state_2_str**(*value*)

ska_tmc_centralnode.model.enum module

class ska_tmc_centralnode.model.enum.**ModesAvailability**(*value*)
Bases: enum.IntEnum

An enumeration.

available = 1

not_available = 0

ska_tmc_centralnode.model.input module

class ska_tmc_centralnode.model.input.**InputParameter**(*changed_callback*)
Bases: object

property tm_subarray_dev_names
Input parameter Return the TM Subarray device names

Returns the TM Subarray device names

Return type tuple

update(*component_manager*)

class ska_tmc_centralnode.model.input.**InputParameterLow**(*changed_callback*)
Bases: *ska_tmc_centralnode.model.input.InputParameter*

property mccs_master_dev_name
Input parameter Return the MCCS Master device name

Returns the MCCS Master device name

Return type str

property mccs_master_leaf_node
Input parameter Return the TM Leaf MCCS Master device name

Returns the TM Leaf MCCS Master device name

Return type str

property mccs_subarray_leaf_node
Input parameter Return the TM Leaf MCCS Subarray device name

Returns the TM Leaf MCCS Subarray device name

Return type str

update(*component_manager*)

class ska_tmc_centralnode.model.input.**InputParameterMid**(*changed_callback*)
Bases: *ska_tmc_centralnode.model.input.InputParameter*

property csp_master_dev_name
Input parameter Return the CSP Master device name

Returns the CSP Master device name

Return type str

property csp_subarray_dev_names
Input parameter Return the CSP Subarray device names

Returns the CSP Subarray device names

Return type tuple

property dish_dev_names
Input parameter Return the dish device names

Returns the TM dish device names

Return type tuple

property sdp_master_dev_name
Input parameter Return the SDP Master device name

Returns the SDP Master device name

Return type str

property sdp_subarray_dev_names
Input parameter Return the SDP Subarray device names

Returns the SDP Subarray device names

Return type tuple

property tm_dish_dev_names
Input parameter Return the TM dish device names

Returns the TM dish device names

Return type tuple

property tm_leaf_csp_master_dev_name
Input parameter Return the CSP Master device name

Returns the CSP Master device name

Return type str

property tm_leaf_sdp_master_dev_name
Input parameter Return the SDP Master device name

Returns the SDP Master device name

Return type str

update(*component_manager*)

ska_tmc_centralnode.model.op_state_model module

Module contents

3.1.2 Submodules

3.1.3 ska_tmc_centralnode.central_node module

Central Node is a coordinator of the complete M&C system. Central Node implements the standard set of state and mode attributes defined by the SKA Control Model.

class ska_tmc_centralnode.central_node.**AbstractCentralNode**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.tmc_base_device.ska_tmc_common.tmc_base_device.TMCBaseDevice._name

Central Node is a coordinator of the complete Telescope system. Central Node is inherited from TMCBaseDevice class which is further inherited from SKABaseDevice class. TMCBaseDevice class contains attributes common to CentralNode and SubarrayNode.

class InitCommand(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.tmc_base_device.TMCBaseDevice.ska_tmc_common.tmc_base_device.TMCBaseDevice.InitCommand._name

A class for the TMC CentralNode's init_device() method.

do()
Initializes the attributes and properties of the Central Node.

Returns A tuple containing a return code and a string message indicating status. The message is for information purpose only.

Return type (ReturnCode, str)

Off()
This command invokes SetStandbyLPMode() command on DishLeafNode, TelescopeOff() command on CspMasterLeafNode and SdpMasterLeafNode.

StandByTelescope()
This command invokes SetStandbyLPMode() command on DishLeafNode, TelescopeStandBy() command on CspMasterLeafNode and SdpMasterLeafNode and TelescopeOff() command on SubarrayNode.

TelescopeOff()
This command invokes SetStandbyLPMode() command on DishLeafNode, Off() command on CspMasterLeafNode and SdpMasterLeafNode.

always_executed_hook()

create_component_manager()

delete_device()

init_command_objects()
Initialises the command handlers for commands supported by this device.

is_AssignResources_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state

Return type boolean

is_Off_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_On_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_ReleaseResources_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_StandByTelescope_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_Standby_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_StartUpTelescope_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_TelescopeOff_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_TelescopeOn_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

is_TelescopeStandby_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

read_desiredTelescopeState()

read_subarrayDevNames()
Return the subarrayDevNames attribute.

read_telescopeHealthState()

read_telescopeState()

read_tmOpState()
Return the tmOpState attribute.

transformedInternalModel_read()

update_command_in_progress_callback(*command_in_progress*)

update_device_callback(*devInfo*)

update_telescope_health_state_callback(*telescope_health_state*)

update_telescope_state_callback(*telescope_state*)

update_tmc_op_state_callback(*tmc_op_state*)

write_subarrayDevNames(*value*)
Set the subarrayDevNames attribute.

3.1.4 ska_tmc_centralnode.central_node_low module

Central Node is a coordinator of the complete M&C system. Central Node implements the standard set of state and mode attributes defined by the SKA Control Model.

class ska_tmc_centralnode.central_node_low.**CentralNodeLow**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.tmc_base_device.ska_tmc_common.tmc_base_device.TMCBaseDevice._name

Central Node is a coordinator of the complete Telescope system

class InitCommand(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.tmc_base_device.TMCBaseDevice.ska_tmc_common.tmc_base_device.TMCBaseDevice.InitCommand._name

A class for the TMC CentralNode's init_device() method.

do()
Initializes the attributes and properties of the Central Node.

Returns A tuple containing a return code and a string message indicating status. The message is for information purpose only.

Return type (ReturnCode, str)

create_component_manager()

init_command_objects()
Initialises the command handlers for commands supported by this device.

read_mccsMasterLeafNodeName()
Return the mccsmasterleafnodename attribute.

read_mccsMasterNodeName()
Return the mccsMasterNodeName attribute.

read_mccsSubarrayLeafNodeName()
Return the mccsSubarrayLeafNodeName attribute.

write_mccsMasterLeafNodeName(*value*)
Set the mccsmasterleafnodename attribute.

write_mccsMasterNodeName(*value*)
Set the mccsMasterNodeName attribute.

write_mccsSubarrayLeafNodeName(*value*)
Set the mccsSubarrayLeafNodeName attribute.

ska_tmc_centralnode.central_node_low.**main**(*args=None*, ***kwargs*)
Runs the CentralNode. :param args: Arguments internal to TANGO
Parameters **kwargs** – Arguments internal to TANGO

Returns CentralNode TANGO object.

3.1.5 ska_tmc_centralnode.central_node_mid module

Central Node is a coordinator of the complete M&C system. Central Node implements the standard set of state and mode attributes defined by the SKA Control Model.

class ska_tmc_centralnode.central_node_mid.**CentralNodeMid**(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.tmc_base_device.ska_tmc_common.tmc_base_device.TMCBaseDevice._name

Central Node is a coordinator of the complete Telescope system

class InitCommand(**args: Any*, ***kwargs: Any*)
Bases: ska_tmc_common.tmc_base_device.TMCBaseDevice.ska_tmc_common.tmc_base_device.TMCBaseDevice.InitCommand._name

A class for the TMC CentralNode's init_device() method.

do()
Initializes the attributes and properties of the Central Node.

Returns A tuple containing a return code and a string message indicating status. The message is for information purpose only.

Return type (ReturnCode, str)

StowAntennas(*argin*)
This command stows the specified receptors.

init_command_objects()
Initialises the command handlers for commands supported by this device.

is_StowAntennas_allowed()
Checks whether this command is allowed to be run in current device state.

Returns True if this command is allowed to be run in current device state.

Return type boolean

read_cspMasterDevName()
Return the cspMasterDevName attribute.

read_cspSubarrayDevNames()
Return the cspsubarraydevnames attribute.

read_dishDevNames()
Return the dishdevnames attribute.

read_imaging()

read_leafCspMasterDevName()
Return the leafcspMasterDevName attribute.

read_leafSdpMasterDevName()
Return the leafsdpMasterDevName attribute.

read_pss()

read_pst()

read_sdpMasterDevName()
Return the sdpMasterDevName attribute.

read_sdpSubarrayDevNames()
Return the sdpsubarraydevnames attribute.

read_tmLeafDishDevNames()
Return the tmleafdishdevnames attribute.

read_vlbi()

update_imaging_callback(*imaging*)

write_cspMasterDevName(*value*)
Set the cspMasterDevName attribute.

write_cspSubarrayDevNames(*value*)
Set the cspsubarraydevnames attribute.

write_dishDevNames(*value*)
Set the dishdevnames attribute.

write_leafCspMasterDevName(*value*)
Set the leafcspMasterDevName attribute.

write_leafSdpMasterDevName(*value*)
Set the leafsdpMasterDevName attribute.

write_sdpMasterDevName(*value*)
Set the sdpMasterDevName attribute.

write_sdpSubarrayDevNames(*value*)
Set the sdpsubarraydevnames attribute.

write_tmLeafDishDevNames(*value*)
Set the tmleafdishdevnames attribute.

ska_tmc_centralnode.central_node_mid.**main**(*args=None*, ***kwargs*)
Runs the CentralNode. :param args: Arguments internal to TANGO
Parameters **kwargs** – Arguments internal to TANGO

Returns CentralNode TANGO object.

3.1.6 ska_tmc_centralnode.dev_factory module

3.1.7 ska_tmc_centralnode.exceptions module

3.1.8 ska_tmc_centralnode.input_validator module

class ska_tmc_centralnode.input_validator.**AssignResourceValidator**(*subarray_list*, *receptor_list*, *dish_prefix*, *logger=<Logger ska_tmc_centralnode.input_validator (WARNING)>*)

Bases: object

Class to validate the input string of AssignResources command of Central Node

loads(*input_string*)

Validates the input string received as an argument of AssignResources command. If the request is correct, returns the deserialized JSON object. The ska-tmc-cdm is used to validate the JSON.

Param input_string: A JSON string

Returns Deserialized JSON object if successful.

Throws InvalidJSONError: When the JSON string is not formatted properly.

SubarrayNotPresentError: If the subarray is not present.

ResourceNotPresentError: When a receptor in the receptor_id_list is not present.

3.1.9 ska_tmc_centralnode.release module

Release information for Python Package

3.1.10 Module contents

CentralNode

Central Node is a coordinator of the complete M&C system.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W