TMC Dish Leaf Nodes Documentation

Release 1.0

NCRA India

Nov 14, 2022

GETTING STARTED

This project is developing the TMC Dish Leaf Nodes component of the Telescope Monitoring and Control (TMC) prototype, for the Square Kilometre Array.

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

CHAPTER

ONE

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.

CHAPTER

TWO

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).

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.

2.2 How to Use

Clone this repo: :: git clone https://gitlab.com/ska-telescope/ska-tmc/ska-tmc-dishleafnode cd ska-tmc-dishleafnode

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

Follow the steps till installation of dependencies then run below command: :: $ 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')

============================= 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)

CHAPTER

THREE

TMC DISH LEAF NODES CODE QUALITY GUIDELINES

3.1 Code formatting / style

3.1.1 Black

TMC Dish Leaf Node 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.

3.1.2 Linting

TMC Dish Leaf Node 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.

3.2 Test coverage

TMC Dish Leaf Node uses pytest to test its code, with the pytest-cov plugin for measuring coverage.

CHAPTER

FOUR

SKA_TMC_DISHLEAFNODE PACKAGE

4.1 Subpackages

4.1.1 ska_tmc_dishleafnode_mid.commands package

Submodules

ska_tmc_dishleafnode_mid.commands.abstract_command module

This module provides base command class for DishLeafNode.

class ska_tmc_dishleafnode.commands.abstract_command.**DishLNCommand**(**args: Any*, ***kwargs: Any*)

Bases: TmcLeafNodeCommand

A base command class for DishLeafNode with the methods and parameters common across all the commands.

init_adapter()

Creates adapter for underlying Dish device.

ska_tmc_dishleafnode_mid.commands.setoperatemode_command module

ska_tmc_dishleafnode_mid.commands.setstandbyfpmode_command module

ska_tmc_dishleafnode_mid.commands.setstandbylpmode_command module

ska_tmc_dishleafnode_mid.commands.setstowmode_command module

Module contents

This is init module for DishLeafNode commands.

class ska_tmc_dishleafnode.commands.**SetOperateMode**(**args: Any*, ***kwargs: Any*)

Bases: *DishLNCommand*

A class for DishLeafNode's SetOperateMode() command.

SetOperateMode invokes SetOperateMode command on Dish Master device.

do(*argin=None*)

Method to invoke SetOperateMode command on DishMaster.

param argin:
None

Returns
(ResultCode, str)

set_operate_mode(*logger*, *task_callback: Optional[Callable] = None*, *task_abort_event: Optional[Event] = None*) → None

A method to invoke the SetOperateMode command. It sets the task_callback status according to command progress.

Parameters

- **logger** (*logging.Logger*) – logger
- **task_callback** (*Callable, optional*) – Update task state, defaults to None
- **task_abort_event** (*Event, optional*) – Check for abort, defaults to None

class ska_tmc_dishleafnode.commands.**SetStandbyFPMode**(**args: Any*, ***kwargs: Any*)

Bases: *DishLNCommand*

A class for DishLeafNode's SetStandbyFPMode() command.

Invokes SetStandbyFPMode (i.e. Full Power State) command on DishMaster.

do(*argin=None*)

Method to invoke SetStandbyFPMode command on DishMaster.

param argin:
None

Returns
(ResultCode, str)

set_standby_fp_mode(*logger*, *task_callback: Optional[Callable] = None*, *task_abort_event: Optional[Event] = None*) → None

A method to invoke the SetStandbyFPMode command. It sets the task_callback status according to command progress.

Parameters

- **logger** (*logging.Logger*) – logger
- **task_callback** (*Callable, optional*) – Update task state, defaults to None
- **task_abort_event** (*Event, optional*) – Check for abort, defaults to None

class ska_tmc_dishleafnode.commands.**SetStandbyLPMode**(**args: Any*, ***kwargs: Any*)

Bases: *DishLNCommand*

A class for DishLeafNode's SetStandbyLPMode() command.

Invokes SetStandbyLPMode (i.e. Low Power State) command on DishMaster.

do(*argin=None*)

Method to invoke SetStandbyLPMode (Low power mode) command on DishMaster.

param argin:
None

Returns
(ResultCode, str)

set_standby_lp_mode(*logger*, *task_callback: Optional[Callable] = None*, *task_abort_event: Optional[Event] = None*)

A method to invoke the SetStandbyLPMode command. It sets the task_callback status according to command progress.

Parameters

- **logger** (*logging.Logger*) – logger
- **task_callback** (*Callable, optional*) – Update task state, defaults to None
- **task_abort_event** (*Event, optional*) – Check for abort, defaults to None

class ska_tmc_dishleafnode.commands.**SetStowMode**(**args: Any*, ***kwargs: Any*)

Bases: *DishLNCommand*

A class for DishleafNode's SetStowMode() command.

SetStowMode command on DishLeafNode enables the telescope to perform further operations and observations. It Invokes SetStowMode command on Dish Leaf Node device.

do(*argin=None*)

Method to invoke SetStowMode command on DishMaster.

param argin:
None

Returns
(ResultCode, str)

set_stow_mode(*logger*, *task_callback: Optional[Callable] = None*, *task_abort_event: Optional[Event] = None*)

A method to invoke the SetStowMode command. It sets the task_callback status according to command progress.

Parameters

- **logger** (*logging.Logger*) – logger
- **task_callback** (*Callable, optional*) – Update task state, defaults to None
- **task_abort_event** (*Event, optional*) – Check for abort, defaults to None

4.1.2 ska_tmc_dishleafnode.manager package

Submodules

ska_tmc_dishleafnode.manager.component_manager module

This module provides an implementation of the Dish Leaf Node ComponentManager.

class ska_tmc_dishleafnode.manager.component_manager.**DishLNComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: TmcLeafNodeComponentManager

A component manager for The Dish Leaf Node component.

is_command_allowed(*command_name: str*) → bool

Checks if the given command is allowed in current operational state.

setoperatemode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetOperateMode command for execution.

Return type
Tuple

setstandbyfpmode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetStandbyFPMode command for execution.

Return type
Tuple

setstandbylpmode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetStandbyLPMode command for execution.

Return type
Tuple

setstowmode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetStowMode command for execution.

Return type
Tuple

Module contents

This is init module for DishLeafNode Manager

class ska_tmc_dishleafnode.manager.**DishLNComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: TmcLeafNodeComponentManager

A component manager for The Dish Leaf Node component.

is_command_allowed(*command_name: str*) → bool

Checks if the given command is allowed in current operational state.

setoperatemode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetOperateMode command for execution.

Return type
Tuple

setstandbyfpmode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetStandbyFPMode command for execution.

Return type
Tuple

setstandbylpmode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetStandbyLPMode command for execution.

Return type
Tuple

setstowmode(*task_callback=None*) → Tuple[ska_tango_base.executor.TaskStatus, str]

Submits the SetStowMode command for execution.

Return type
Tuple

4.2 Submodules

4.3 ska_tmc_dishleafnode._dish_leaf_node module

This is DishLeafNode TANGO device.

class ska_tmc_dishleafnode.dish_leaf_node.**DishLeafNode**(**args: Any*, ***kwargs: Any*)

Bases: SKABaseDevice

A Leaf control node for DishMaster.

Device Properties

DishMasterFQDN:
FQDN of Dish Master Device

Device Attributes

commandExecuted:
Stores command executed on the device.

dishMasterDevName:
Stores Dish Master Device name.

class InitCommand(**args: Any*, ***kwargs: Any*)

Bases: InitCommand

A class for the TMC DishLeafNode init_device() method.

do()

Initializes the attributes and properties of the DishLeafNode.

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

rtype:
(ResultCode, str)

create_component_manager()

delete_device()

init_command_objects()

Initialises the command handlers for commands supported by this device.

is_Abort_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_Configure_allowed()

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

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

device state.

Return type
boolean

`is_EndScan_allowed()`

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

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

device state.

Return type
boolean

`is_ObsReset_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_Restart_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 :rtype: boolean

`is_Scan_allowed()`

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

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

device state.

Return type
boolean

`is_SetOperateMode_allowed()`

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

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

device state.

Return type
boolean

`is_SetStandbyFPMode_allowed()`

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

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

device state.

Return type
boolean

`is_SetStandbyLPMode_allowed()`

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

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

device state.

Return type
boolean

`is_SetStowMode_allowed()`

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

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

device state.

Return type
boolean

`is_StartCapture_allowed()`

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

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

device state.

Return type
boolean

`is_StopCapture_allowed()`

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

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

device state.

Return type
boolean

`is_StopTrack_allowed()`

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

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

device state.

Return type
boolean

`is_Track_allowed()`

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

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

device state.

Return type
boolean

read_dishMasterDevName()

Returns the dishMasterDevName attribute value.

write_dishMasterDevName(*value*)

Set the dishMasterDevName attribute.

ska_tmc_dishleafnode.dish_leaf_node.**main**(*args=None*, ***kwargs*)

Runs the DishLeafNode. :param args: Arguments internal to TANGO :param kwargs: Arguments internal to TANGO :return: DishLeafNode TANGO object.

4.4 Module contents

This is a DishLeafNode package which is responsible for monitoring and control of the Dish device.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search
- search

PYTHON MODULE INDEX

S

INDEX

M

R

S

W