developer.skatelescope.org Documentation

SKA Organization

Sep 08, 2023

BACKGROUND

CHAPTER

ONE

OVERVIEW

The Mid.CBF Master Control Software (MCS) provides a high-level interface to the Telescope Monitoring and Control (TMC) and CSP_Mid Local Monitoring and Control (LMC), and translates the high-level commands into the configuration and control of individual Talon-DX boards.

1.1 System Context

The following diagram shows MCS as it fits into the rest of the CSP Mid system.

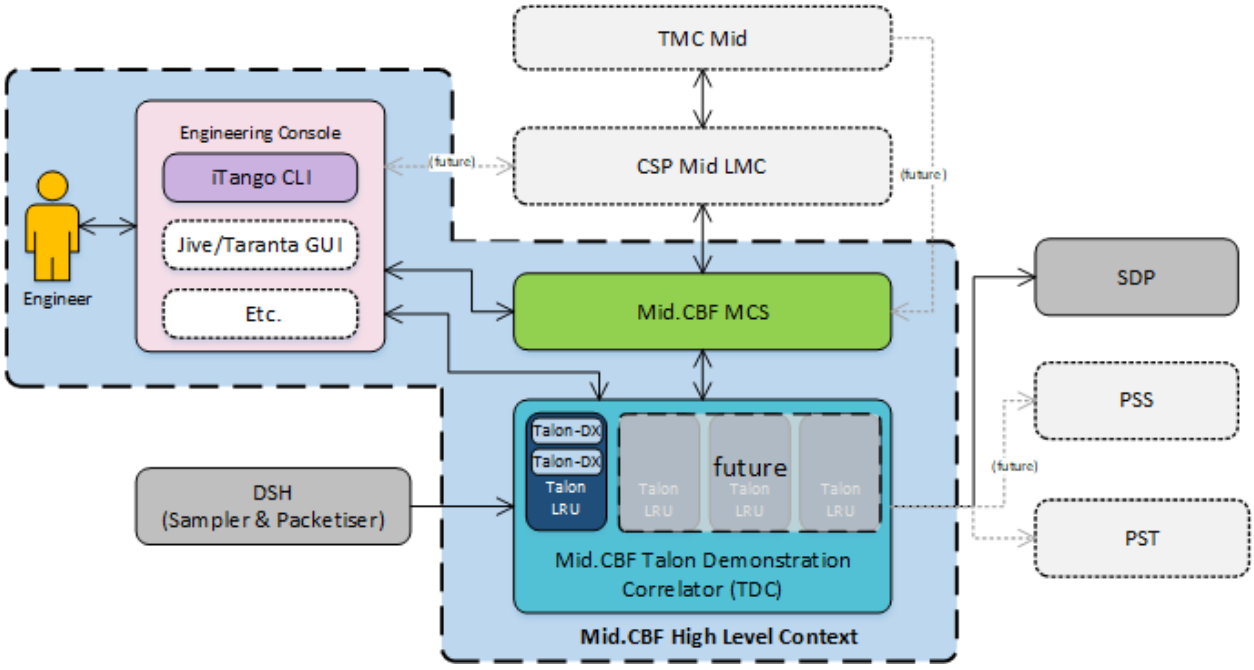
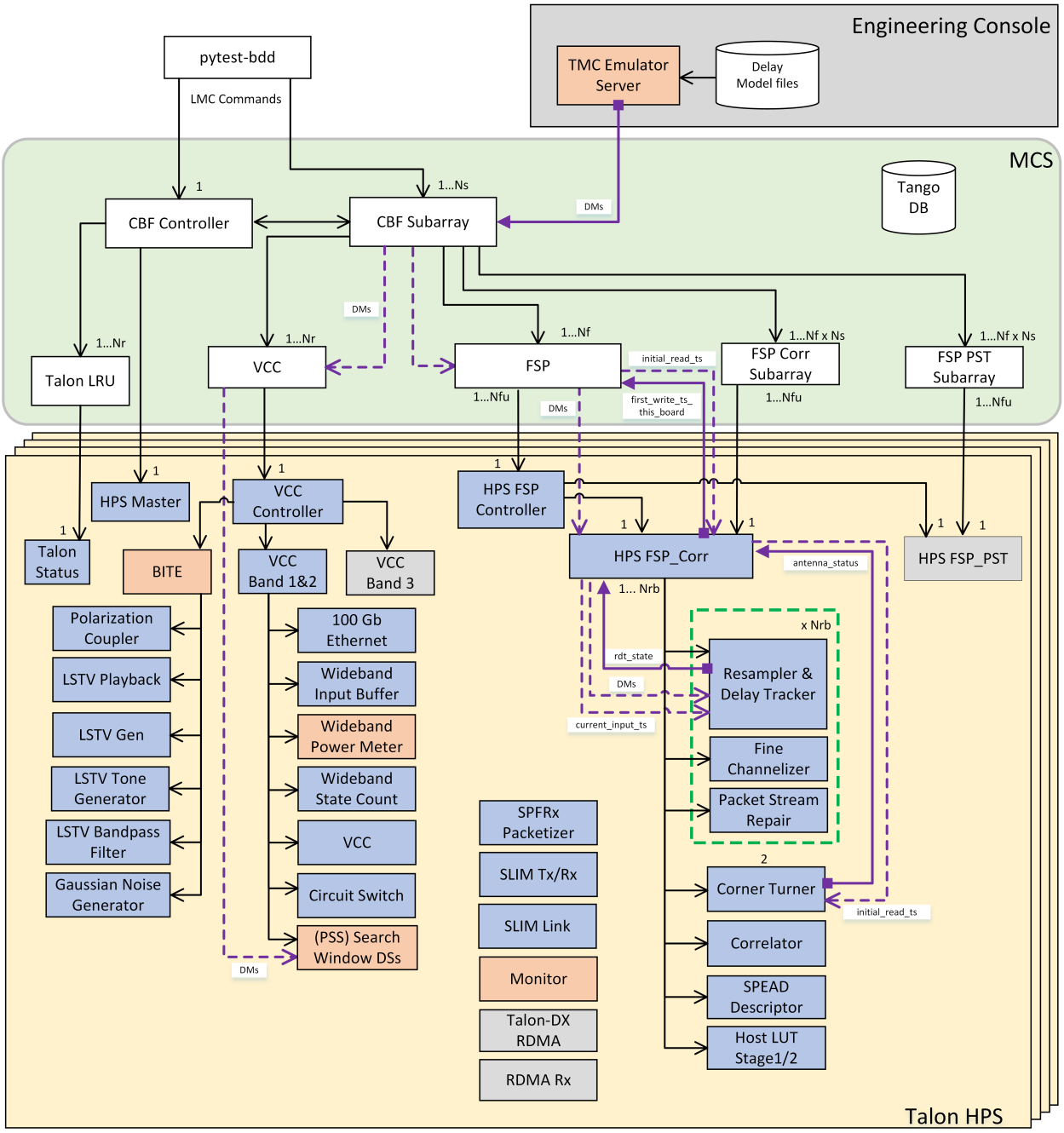


Fig. 1: MCS System Context

MCS interacts with TMC and CSP_Mid LMC and controls the applications and low-level device servers that run on the hard processor system (HPS) of the Talon-DX boards. The diagram below shows these interactions. TMC and CSP_Mid LMC interact with the CBF Controller and CBF Subarray. The diagram also shows the Engineering Console and pytest-BDD elements which are used to test the interfaces between MCS and TMC/CSP_Mid LMC during development.

In MCS there is one VCC device and one FSP device for each VCC/FSP in the system. These devices communicate directly with the top level HPS application for the reconfiguration of the Talon-DX functionality (VCC Controller / HPS FSP Controller)



Nr = Max number of Receptors (= 4*)
Nf = Max number of FSPs (=4*)
Ns = Max number of Subarrays (=2*)
Nfu = Number of FPGAs in an FSP-UNIT
Nrb = Number of receptor data streams per board
* : Current MCS values
Note: for TDC, Nr = Nf, Nrb = 8, Nfu = 1

Publish-Subscribe
Data sent on callback triggered by event
Command/Attribute
Per-receptor processing
Device Server
Disabled Device Server
Not currently implemented

CHAPTER

TWO

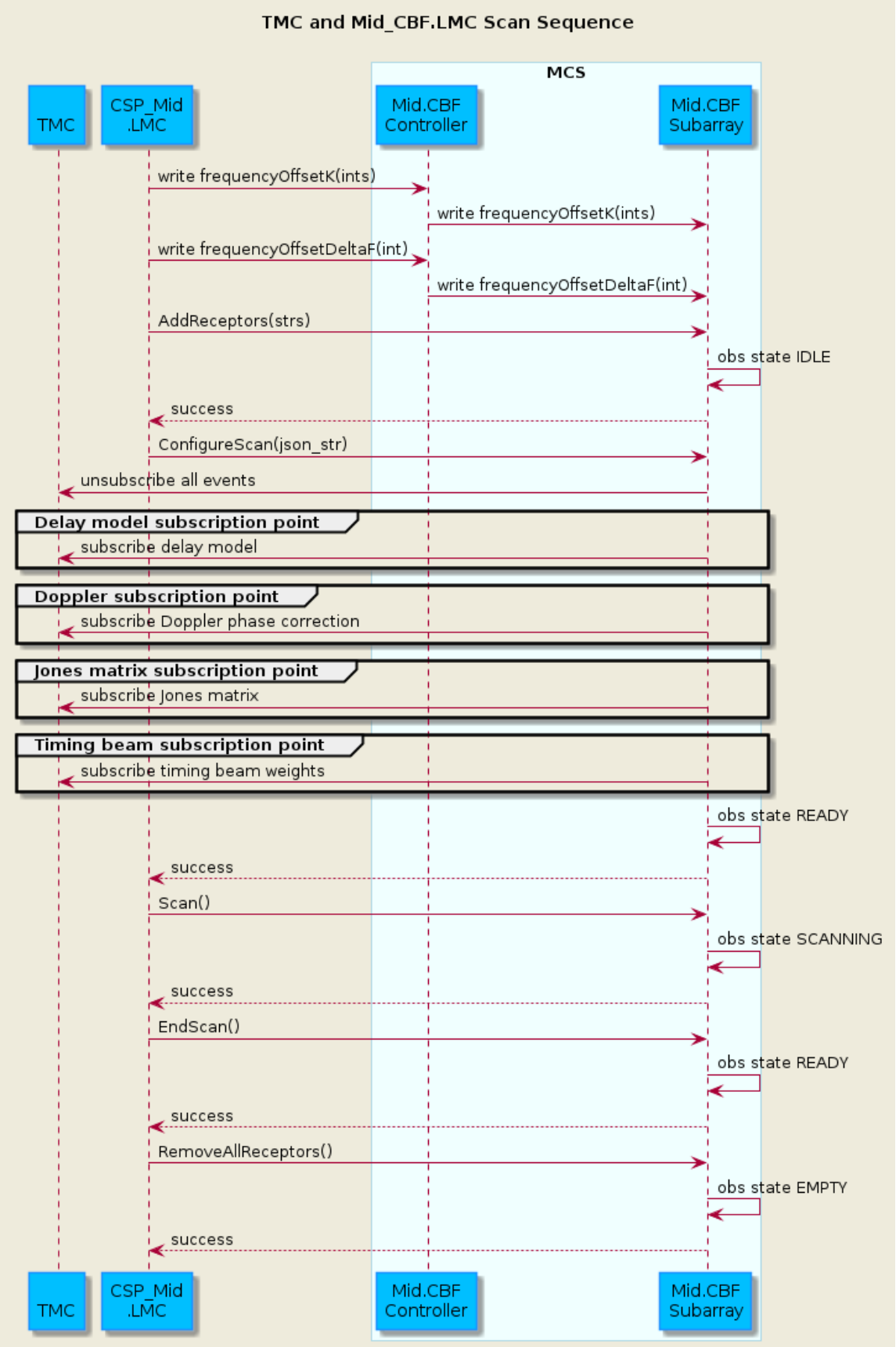
LMC TO MCS

MCS provides commands and attributes to turn MCS on and off (through the CBF Controller) as well as commands needed to configure and execute scans through the subarrays. (CBF Subarray)

The sequence diagram below shows the interactions between LMC and MCS to assign receptors to a subarray, configure a scan, and run a scan. It shows configuration of one Mid.CBF subarray followed by running a scan on that subarray. It ends with no receptors assigned to the subarray. The calls to write the frequency offset K and frequency offset delta F values only need to be written when there are updates to the values. They must be written to the CBF Controller before the scan configuration.

For full details of MCS Controller see *CbfController*.

For full details of MCS Subarray see *CbfSubarray*.


TMC and Mid_CBF.LMC Scan Sequence
MCS
TMC
CSP_Mid .LMC
Mid.CBF Controller
Mid.CBF Subarray
write frequencyOffsetK(ints)
write frequencyOffsetK(ints)
write frequencyOffsetDeltaF(int)
write frequencyOffsetDeltaF(int)
AddReceptors(strs)
obs state IDLE
success
ConfigureScan(json_str)
unsubscribe all events
Delay model subscription point
subscribe delay model
Doppler subscription point
subscribe Doppler phase correction
Jones matrix subscription point
subscribe Jones matrix
Timing beam subscription point
subscribe timing beam weights
obs state READY
success
Scan()
obs state SCANNING
success
EndScan()
obs state READY
success
RemoveAllReceptors()
obs state EMPTY
success
TMC
CSP_Mid .LMC
Mid.CBF Controller
Mid.CBF Subarray

2.1 CbfController Tango Commands

Command	Parameters	Return type	Action
Off	None	(ResultCode, str)	Turn off the controller and subordinate devices
Standby	None	(ResultCode, str)	Put in low power mode
On	None	(ResultCode, str)	Turn on the controller and subordinate devices

2.2 CbfSubarray Tango Commands

Command	Parameters	Return type	Action
Abort	None	(ResultCode, str)	Abort subarray configuration or operation
AddReceptors	List[str]	(ResultCode, str)	Assign receptors to this subarray Turn subarray to ObsState = IDLE if no receptor was previously assigned
ConfigureScan	JSON str*	(ResultCode, str)	Change state to configuring Configure attributes from input JSON Subscribe events Configure VCC, VCC subarray, FSP, FSP Subarray. Publish output links.
EndScan	None	(ResultCode, str)	End the scan
ObsReset	None	(ResultCode, str)	Reset subarray scan configuration
Off	None	(ResultCode, str)	Set subarry power mode to off. Commands FSP<function mode> Subarrays to turn off
On	None	(ResultCode, str)	Set subarry power mode to on. Commands FSP<function mode> Subarrays to turn on
RemoveAllReceptors	None	(ResultCode, str)	Remove all receptors Turns Subarray off if no receptors are assigned
RemoveReceptors	List[str]	(ResultCode, str)	Remove receptors in input list Turns Subarray to ObsState=EMPTY if no receptors assigned

- Schema for JSON string defined in the Telescope Model - Mid.CBF schemas

CHAPTER

THREE

MCS TO HPS

The interface from the MCS to the HPS is largely in the form of communication between Tango devices running on either side.

The interface also currently consists of low-level SSH calls from the MCS to the Talon-DX boards, which are used to copy FPGA bitstreams and Tango device server binaries to the boards and start the HPS Master process. This functionality may be moved in the future, but for now it is implemented in the *TalonDxComponentManager Class*, which is instantiated by the *CbfController*.

3.1 MCS and HPS Master DS

The interface between the MCS and the HPS Master device server is primarily made up of the `configure` command sent from the MCS to the HPS master, which programs the FPGA and spawns the remaining HPS device servers. Before this command can be run, it is expected that the MCS has already copied the necessary bitstreams and binaries to the board and the HPS master has obviously been started. This is all handled automatically as part of the *MCS On Command*.

The `configure` command has one argument, which is a JSON-formatted string. An example of its contents can be seen below.

```
{
    "description": "Configures Talon DX to run VCC firmware and devices.",
    "target": "talon1",
    "ip_address": "169.254.100.1",
    "ds_hps_master_fqdn": "talondx-001/hpsmaster/hps-1",
    "fpga_path": "/lib/firmware",
    "fpga_dtb_name": "vcc3_2ch4.dtb",
    "fpga_rbf_name": "vcc3_2ch4.core.rbf",
    "fpga_label": "base",
    "ds_path": "/lib/firmware/hps_software/vcc_test",
    "server_instance": "talon1_test",
    "devices": [
        "dscircuitswitch",
        "dsdct",
        "dsfinechannelizer",
        "dstalondxrdma",
        "dsvcc"
    ]
}
```

3.2 MCS On Command

The following diagram shows the `CbfController` On command sequence and how it integrates with other components in the Mid.CBF system. The steps are outlined in detail in the Engineering Console.

From a MCS perspective, the On command sequence consists of the following steps:

- Arrows 4-7: Power on the Talon-DX boards
- Arrow 9: Attempt to connect to each board over SSH (see *TalonDxComponentManager Class*)
- Arrows 8-9: Copy the relevant binaries and bitstreams to each board
- Arrow 10: Start up the HPS Master on each board
- Arrow 12: Send the `configure` to each HPS Master device server

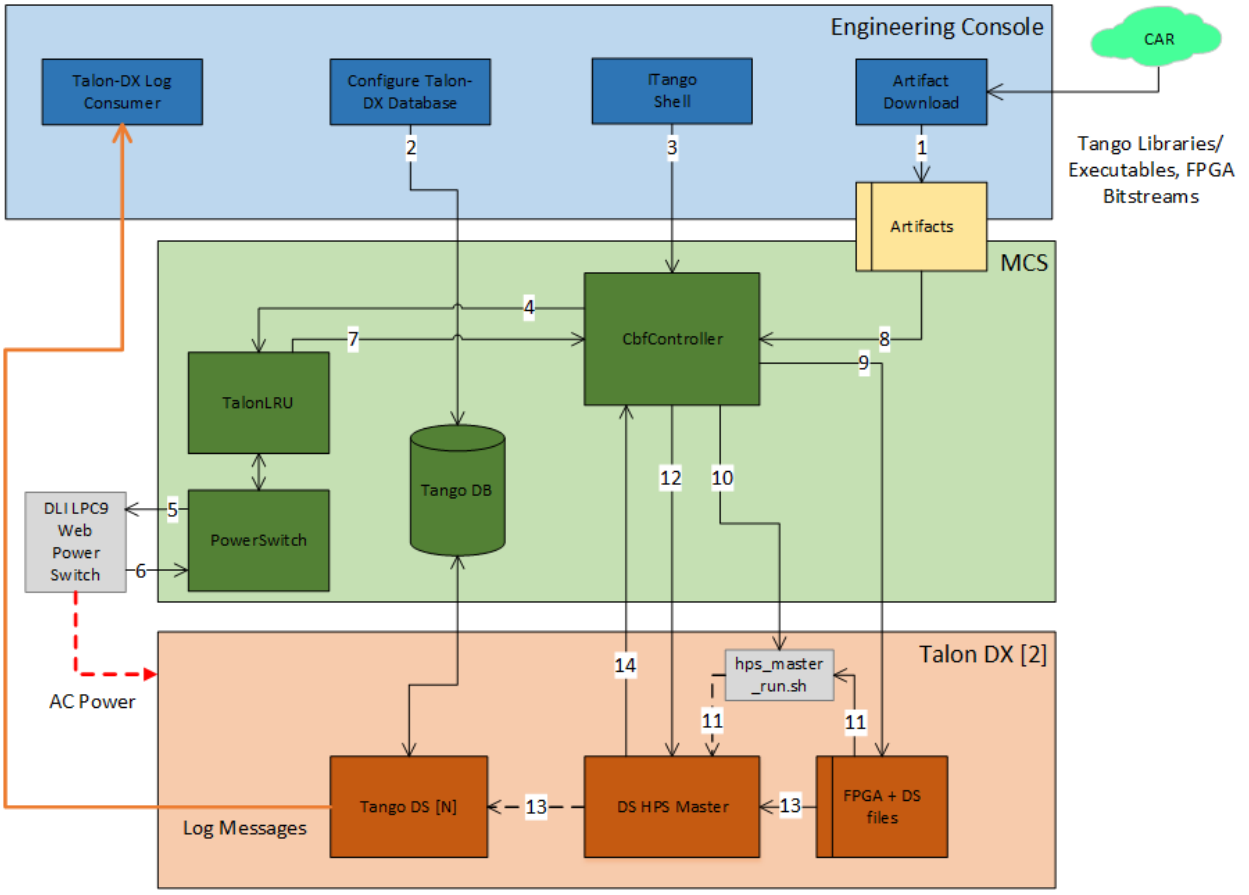
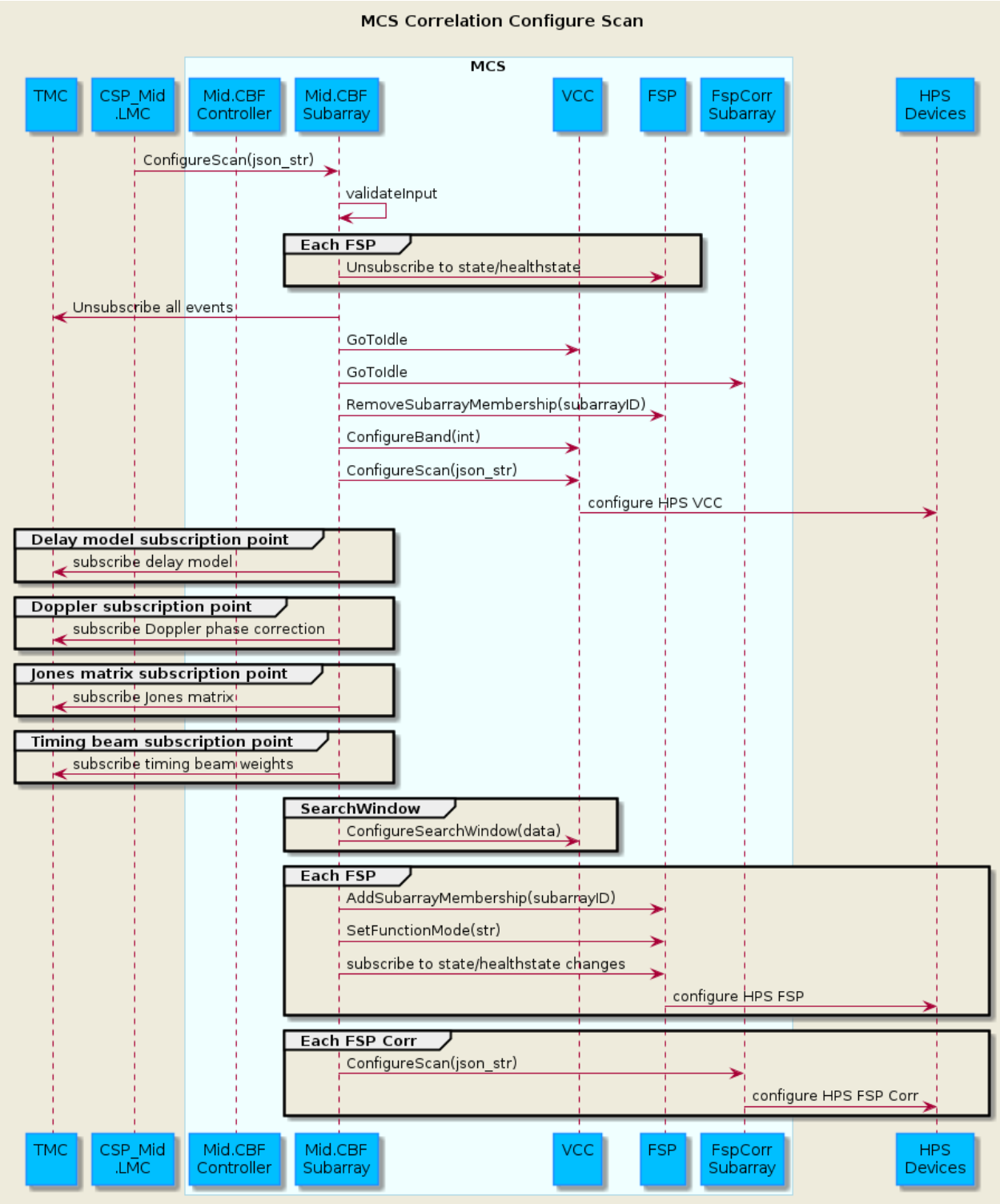


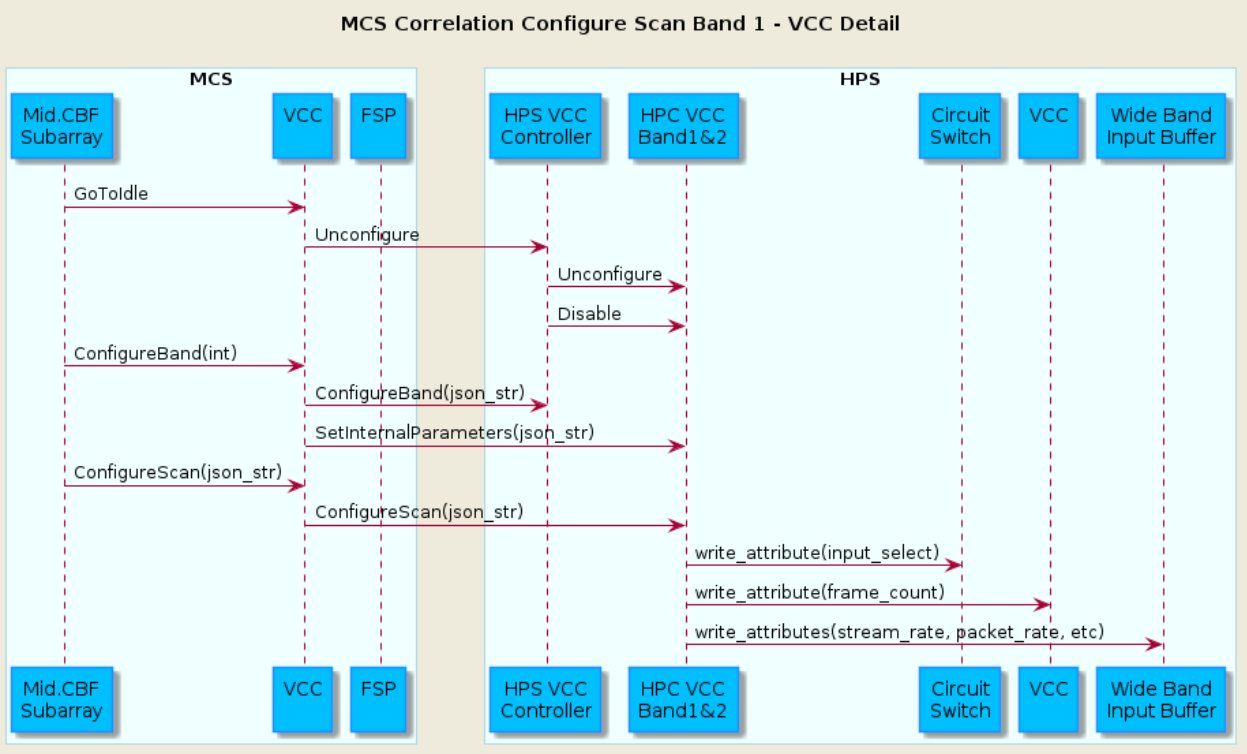
Fig. 1: MCS On Command Sequence

3.3 Configure Scan Command Sequence

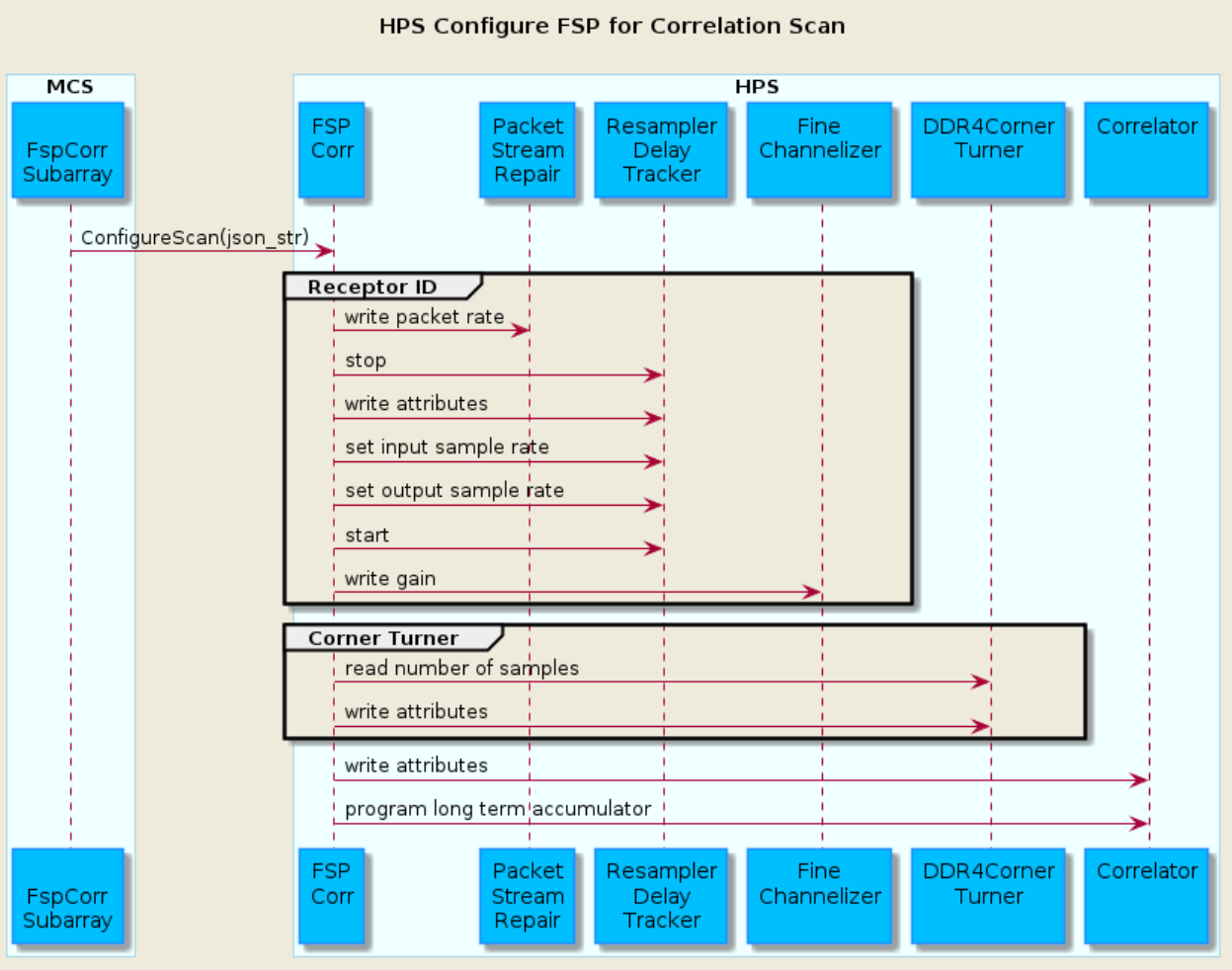
The sequence diagram below shows the main sequence of calls in MCS to configure a correlation scan. Return calls are not shown.



The sequence diagram below shows additional detail for configuration of the VCC for a correlation scan.



The sequence diagram below shows details of calls to configure a FSP for a correlation scan.


HPS Configure FSP for Correlation Scan
MCS
HPS
FspCorr Subarray
FSP Corr
Packet Stream Repair
Resampler Delay Tracker
Fine Channelizer
DDR4Corner Turner
Correlator
ConfigureScan(json_str)
Receptor ID
write packet rate
stop
write attributes
set input sample rate
set output sample rate
start
write gain
Corner Turner
read number of samples
write attributes
write attributes
program long term accumulator
FspCorr Subarray
FSP Corr
Packet Stream Repair
Resampler Delay Tracker
Fine Channelizer
DDR4Corner Turner
Correlator

CHAPTER

FOUR

GETTING STARTED

4.1 Developer Setup

This section follows the instructions on the SKA developer's portal:

- Dev Environment Setup
- Dev FAQs

4.2 Git Repository

The MCS Git Repository is available at the following page: https://gitlab.com/ska-telescope/ska-mid-cbf-mcs

The README on the repository will guide users through cloning and initializing the repository.

4.3 Running the Mid CBF MCS

The ska-mid-cbf-mcs Tango device servers are started and run via Kubernetes.

Make sure Kubernetes/minikube and Helm have been installed (and verified) as described in the 'Set up Kubernetes' section.

Note: You may need to change permission to the .minikube and .kube files in your home directory:

```
sudo chown -R <user_name>:<user_name> ~/.minikube/
sudo chown -R <user_name>:<user_name> ~/.kube/
```

4.3.1 1. Make sure minikube is up and running

The following commands use the default minikube profile. If you are running the MCS on the Dell server you will need to set up your own minikube profile. A new minikube profile is needed for a new cluster, so creating minikube profiles ensures two users on the Dell server do not work on the same cluster at the same time. To view the modifications needed to run the following commands on a new minikube profile see Minikube Profiles

```
minikube start    # start minikube (local kubernetes node)
minikube status   # check current status of minikube
```

The `minikube status` output should be:

```
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

If restarting a stopped minikube; from local `ska-cicd-deploy-minikube` repository run `make minikube-metallb-config` to reapply metallb configMap to determine pod LoadBalancer service external IP addresses.

4.3.2 2. From the root of the project, build the application image.

```
cd ska-mid-cbf-mcs
eval $(minikube docker-env)   # to use the minikube's docker environment
make oci-image-build          # if building from local source and not artefact repository
```

`make oci-image-build` is required only if a local image needs to be built, for example every time the SW has been updated. For development, in order to get local changes to build, run `eval $(minikube docker-env)` before `make build`

Note: To check if you are within the minikube's docker environment, use the `minikube status` command. It will indicate `docker-env: in use` if in use as follows:

```
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
docker-env: in-use
```

4.3.3 3. Install the umbrella chart.

```
make k8s-install-chart        # deploy from Helm charts
make k8s-install-chart-only   # deploy from Helm charts without updating dependencies
```

Note: `make k8s-watch` will list all of the pods' status in every 2 seconds using kubectl; `make k8s-wait` will wait until all jobs are 'Completed' and pods are 'Running'.

4.3.4 4. (Optional) Create python virtual environment to isolate project specific dependencies from your host environment.

```
virtualenv venv           # create python virtualenv 'venv'
source venv/bin/activate  # activate venv
```

4.3.5 5. Install linting and testing requirements.

```
make requirements
```

4.3.6 6. Install the MCS package in editable mode.

```
pip install -e .
```

4.3.7 7. Run a test.

```
make k8s-test  # functional tests with an already running deployment
make python-test  # unit tests, deployment does not need to be running
```

Note: add `-k` pytest flags in `setup.cfg` in the project root to limit which tests are run

4.3.8 8. Tear down the deployment.

```
make k8s-uninstall-chart              # uninstall deployment from Helm charts
deactivate                            # if in active virtualenv
eval $(minikube docker-env --unset)   # if docker-env variables were set previously
minikube stop                         # stop minikube
```

4.4 Useful Minikube Commands

4.4.1 Create a minikube

```
minikube start
```

4.4.2 Check the status of the cluster created for your minikube

```
minikube status
```

4.4.3 Fixing a Misconfigured Kubeconfig

If the kubeconfig is pointing to a stale minikube and is showing as `Misconfigured` when checking the `minikube status`, or if the minikube's IP or port has changed, update the context as follows:

```
minikube update-context
```

4.4.4 Delete a minikube profile

```
minikube delete
```

4.4.5 Set and unset docker-env variables

```
eval $(minikube docker-env)
eval $(minikube docker-env --unset)
```

4.5 Taranta

This provides a graphical user interface using Taranta (previously known as WebJive); to set it up:

- Add the following line to `/etc/hosts`:

  ```
  192.168.49.2  taranta
  ```

  Note: 192.168.49.2 is the minikube IP address, obtainable with the command `minikube ip`

- Navigate to `taranta/ska-mid-cbf/taranta/devices` in a browser (works best with Google Chrome).

The following credentials can be used to operate the system:

- Username: `user1`
- Password: `abc123`

The device tree can be viewed and explored. In addition, device attributes can be seen and modified, and device commands can be sent, by creating and saving a new dashboard.

4.6 Generating Documentation

To re-generate the documentation locally prior to checking in updates to Git:

```
make docs-build html
```

To see the generated documentation, open `/ska-mid-cbf-mcs/docs/build/html/index.html` in a browser – e.g.,

```
firefox docs/build/html/index.html &
```

4.7 Releasing

For a new release (i.e. prior to merging a branch into main) update the following files by incrementing version/release/tag number fields to conform to the semantic versioning convention:

- `.release`: `release=` and `tag=`
- `src/ska_mid_cbf_mcs/release.py`: `version =`
- `charts/ska-mid-cbf/Chart.yaml`: `version:` and `appVersion:`
- `charts/ska-mid-cbf/values.yaml`: `midcbf:image:tag:`

- `charts/ska-mid-cbf-tmleafnode/Chart.yaml`: `version:` and `appVersion:`
- `charts/ska-mid-cbf-tmleafnode/values.yaml`: `midcbf:image:tag:`
- `charts/mid-cbf-umbrella/Chart.yaml`:
  - `version:` and `appVersion:`
  - `version:` under `ska-mid-cbf` and `ska-mid-cbf-tmleafnode`

Note: `appVersion` represents the version of the application running, so it corresponds to the ska-mid-cbf-mcs docker image version.

Once a new release has been merged into main, create a new tag on GitLab and run the manual "publish-chart" stage of the tag pipeline to publish the Helm charts.

4.8 Development resources

4.8.1 Other resources

See more tango device guidelines and examples in the `ska-tango-examples` repository

4.8.2 Useful commands

Kubernetes

For Kubernetes basic kubectl commands see: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

To display components of the MCS Kubernetes system:

```
kubectl get all -n ska-mid-cbf
kubectl describe <component-name> -n ska-mid-cbf  # info on a particular component
```

This should list the following running pods:

- `cbfcontroller-controller-0` : The `CbfController` TANGO device server.
- `cbfsubarrayxx-cbf-subarray-xx-0`: `xx` ranges from `01` to `03`. The 3 instances of the `CbfSubarray` TANGO device server.
- `fspxx-fsp-xx-0`: `xx` ranges from `01` to `04`. The 4 instances of the `FspMulti` TANGO device servers.
- `vccxxx-vcc-xxx-0`: `xxx` ranges from `001` to `004`. The 4 instances of the `VccMulti` TANGO device servers.
- `tmcspsubarrayleafnodetestx-tmx-0`: `x` ranges from `1` to `2`. The 2 instances of the `TmCspSubarrayLeafNodeTest` TANGO device servers.
- `tango-host-databaseds-from-makefile-test-0`: The TANGO DB device server.
- etc.

4.9 License

See the `LICENSE` file for details.

CHAPTER

FIVE

CODE ELEMENT DESIGN NOTES

5.1 Component Managers

More details about the role of component managers can be found in the SKA Tango Base Documentation. In the Mid.CBF MCS each component has a Tango device class and a component manager class. The Tango device class updates its state model(s) (the `op_state_model` andor `obs_state_model`). The Tango device class does not directly communicate with its component, instead it tells its component manager class what to do by calling its methods. The component manager class directly interacts with its component. Its role is to establish communication with its component and monitor and control it. An example of this Tango device and component manager interaction is shown in the diagram below.

5.2 Cbf Controller

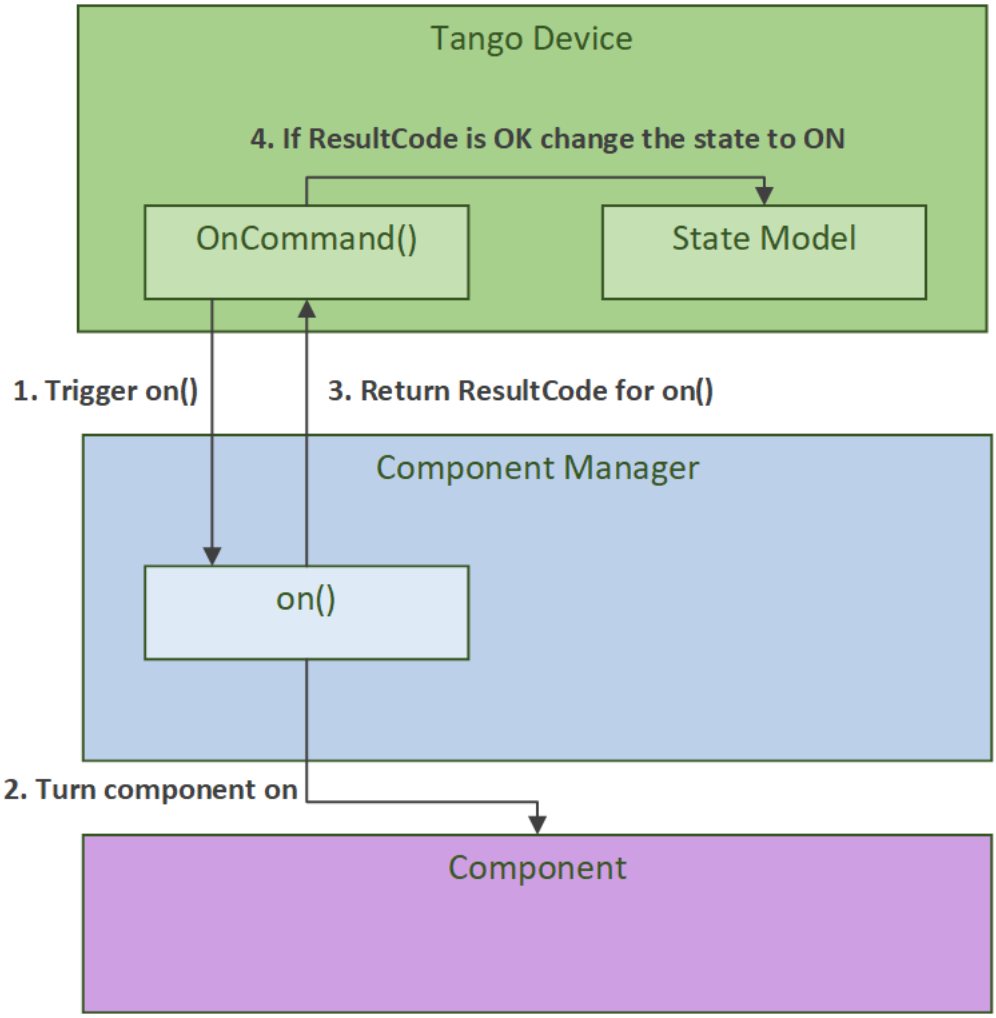
The `CbfController` Tango device controls its subordinate Tango devices: `Fsp`, `Vcc`, `CbfSubarray` and `TalonLRU`. It is responsible for turning these subordinate devices on and off, and putting the `Fsp`,``Vcc`` and CbfSubarray devices in STANDBY mode. The CbfController also initiates the configuration of the Talon-DX boards. The `CbfController` device's OnCommand triggers `TalonDxComponentManager.configure_talons` to be called which copies the device server binaries and FPGA bitstream to the Talon-DX boards, starts the HPS master device server and sends the configure command to each DsHpsMaster device.

5.3 Cbf Subarray

The `CbfSubarray` Tango device is used to monitor and control scan operation of a Mid.CBF receptor subarray. This device receives one configuration per scan, and a subarray may accept this scan configuration only after being assigned at least one receptor.

5.3.1 Receptor assignment

Receptor assignment to a subarray is done before configuration for a scan. Receptor assignment is exclusive; receptors assigned to one subarray cannot belong to any other subarray. Up to 197 receptors can be assigned to one subarray; currently, there is only support for 4 receptors.


Tango Device
4. If ResultCode is OK change the state to ON
OnCommand()
State Model
1. Trigger on()
3. Return ResultCode for on()
Component Manager
on()
2. Turn component on
Component

5.3.2 Scan configuration

Subarrays receive a scan configuration via an ASCII encoded JSON string. The scan configuration is validated for completeness and its parameters implemented as Tango device attributes; the subarray device will then also configure subordinate devices with the relevant parameters, including VCC, FSP and FSP-subarray devices.

5.4 Frequency Slice Processor (FSP)

The `Fsp` Tango device is used for monitoring and control of a Frequency Slice Processor (FSP) during scan operation. An FSP device can be configured for processing of one of up to twenty-six frequency slices (depending on observational frequency band). Additionally, an FSP can be assigned to any number of subarrays with matching configurations.

5.4.1 Fsp Function Mode

There are four function modes available for FSP scan configuration, each with a corresponding function mode capability and subarray device per FSP; furthermore, each FSP function mode subarray device corresponds to a unique pairing of one FSP with one subarray. Currently, one subarray and four FSPs are supported.

FSP Function Mode Subarray devices:

- Correlation (CORR): `FspCorrSubarray`
- Pulsar Search Beamforming (PSS-BF): `FspPssSubarray`
- Pulsar Timing Beamforming (PST-BF): `FspPstSubarray`
- VLBI Beamforming (VLBI): `FspVlbiSubarray`

5.5 Mid.Cbf VCC Device Server (VccMulti)

5.5.1 VCC Device

The `Vcc` Tango device is used to control and monitor the functionality for a single Talon-DX board that runs VCC functionality. This device communicates with the top-level VCC device server running on the Talon-DX board to coordinate setup and processing activites of low-level device servers.

The `Vcc` device can operated in either simulation mode or not. When in simulation mode (this is the default), simulator classes are used in place of communication with the real Talon-DX Tango devices. This allows testing of the MCS without any connection to the hardware.

5.6 Talon LRU

The `TalonLRU` Tango device handles the monitor and control functionality for a single Talon LRU. A TalonLRU instance must therefore be created for each LRU. Currently this device only controls the power to the LRU via a proxy to the `PowerSwitch` device.

The operational state of this device always reflects the power state of the LRU. If at least one of the PDU outlets connected to the LRU is switched on, the state of the `TalonLRU` device should be ON. If both outlets are switched off, then the state should be OFF.

If the state of the outlets is not consistent with the state of the `TalonLRU` device when it starts up (or when `simulationMode` of the `PowerSwitch` device changes), the `TalonLRU` device transitions into a FAULT state. The

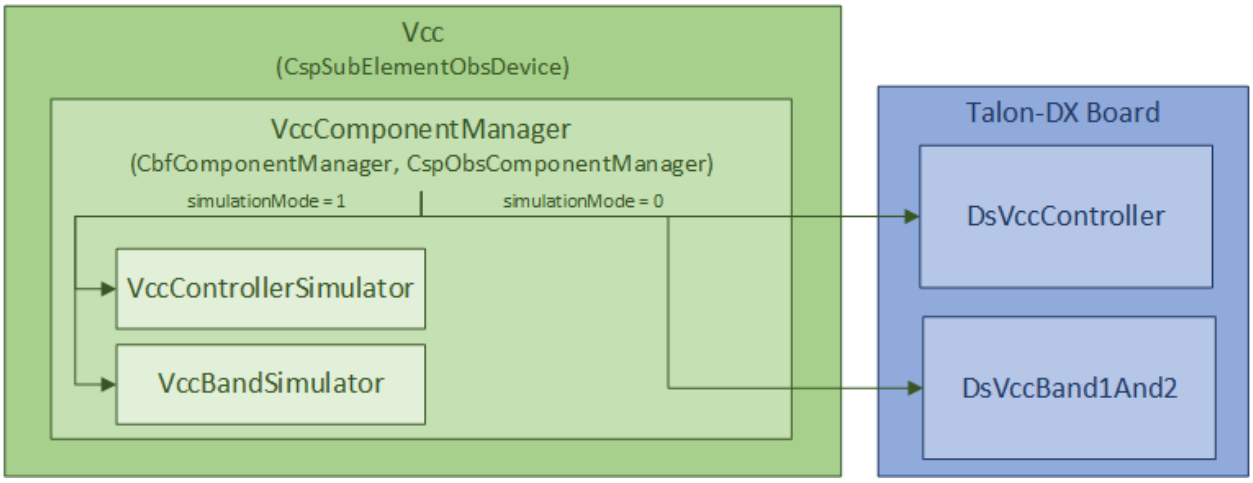


Fig. 1: MCS Vcc Device

power outlets must then be manually switched to the expected startup state via some other method before resetting the `TalonLRU` device.

The expected startup state of the device is OFF.

5.7 Power Switch

The `PowerSwitch` Tango device is used to control and monitor the web power switch that provides power to the Talon LRUs. The current power switch in use is the DLI LPC9 (User Guide). The power switch has 8 programmable outlets, meaning that it can power up to 4 Talon LRUs (each LRU needs two power supply lines).

The `PowerSwitch` device can be operated in either simulation mode or not. When in simulation mode (this is the default), the `PowerSwitchSimulator` is used in place of communication with the real power switch hardware. This allows testing of the MCS with no hardware connected.

When integration testing with the hardware is desired, the `simulationMode` attribute can be set to 0. This initializes communication with the real power switch via the `PowerSwitchDriver`, and queries the list of outlets in the power switch.

Important operational notes:

- Certain requests to the power switch hardware can take longer than others, hence a timeout of 4 seconds set in the `PowerSwitchDriver`. As such, accessing attributes or commands in the `PowerSwitch` device can take longer than the default Tango timeout (3 seconds). Any `DeviceProxy` of the `PowerSwitch` device should increase its timeout to 5 seconds to safely complete all requests (both successful and unsuccessful) before the Tango timeout. This can be done using `pwr_dev_proxy.set_timeout_millis(5000)`, assuming `pwr_dev_proxy` is a `DeviceProxy` to the `PowerSwitch` device.
- Although the DLI LPC9 claims to support up to 8 concurrent clients, testing has shown a significant slow down in response time when more than one request has been sent to the power switch. As such, all communication with the power switch should be kept sequential. Currently the `PowerSwitchDriver` does not ensure this. If the `PowerSwitch` device is ever changed to handle requests asynchronously, the `PowerSwitchDriver` should also be updated to only process one request at a time.

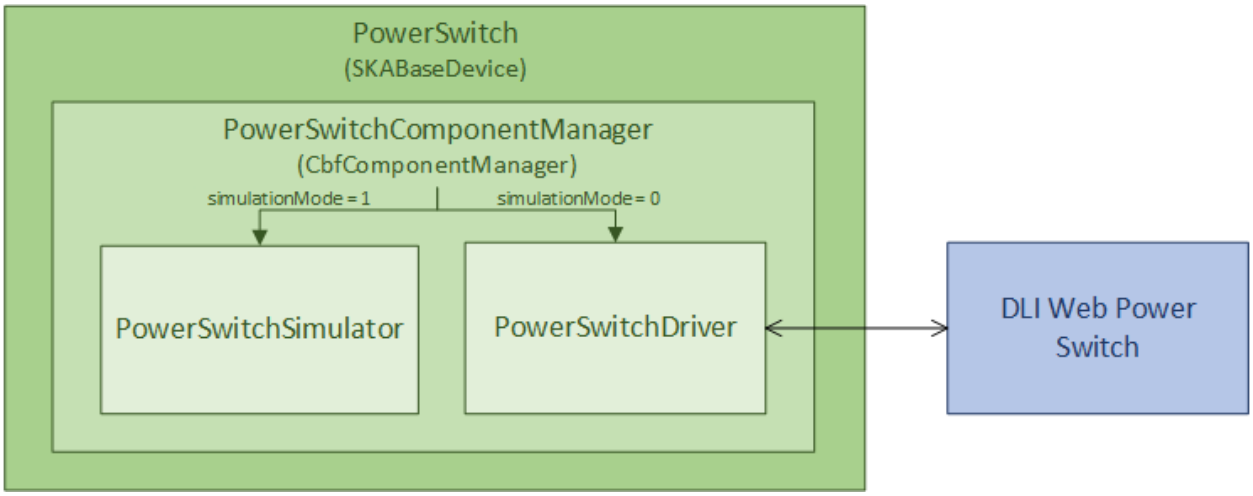


Fig. 2: MCS PowerSwitch Device

5.8 Talon DX Log Consumer

The Talon DX Log Consumer is a Tango device intended to run on the host machine that connects to the Talon-DX boards. This Tango device is set up as a default logging target for all the Tango device servers running on the HPS of each Talon-DX board. When the HPS device servers output logs via the Tango Logging Service, the logs get transmitted to this log consumer device where they get converted to the SKA logging format and outputted once again via the SKA logging framework. In this way logs from the Talon-DX boards can be aggregated in once place and eventually shipped to the Elastic framework in the same way as logs from the Mid CBF Monitor and Control Software (MCS).

Note: more instances of the device may be created to provide enough bandwidth for all the HPS device servers.

5.8.1 Connecting from HPS DS to the Log Consumer

The Talon-DX boards connect to the host machine (currently known as the development server) over a single Ethernet connection. The IP address of the development server on this connection is `169.254.100.88` and all outgoing traffic from the Talon-DX boards must be addressed to this IP.

When the log consumer starts up on the development server, the OmniORB end point (IP address and port) it is assigned is local to the development server (i.e. IP address `142.73.34.173`, arbitrary port). Since the Talon boards are unable to connect to this IP address. we need to manually publish a different endpoint when starting up the log consumer that is visible to the HPS devices.

The following ORB arguments are used (see the make target `talondx-log-consumer`):

- `-ORBendPointPublish giop:tcp:169.254.100.88:60721`: Exposes this IP address and port to all clients of this Tango device. When the HPS device servers contact the database to get the network information of the log consumer, this is the IP address and port that is returned. The IP addresses matches that of the Ethernet connection to the development server, allowing the HPS device servers to direct their messages across that interface.
- `-ORBendPoint giop:tcp:142.73.34.173:60721`: Assigns the IP address and port that the log consumer device is actually running on. This needs to be manually assigned since an iptables mapping rule was created on the development server to route any TCP traffic coming in on `169.254.100.88:60721` to `142.73.34.173:60721`.

Some important notes:

- Due to the end point publishing, no Tango devices running on the development server will be able to connect to the log consumer (including being able to configure the device from Jive). This is because the published IP address is not accessible on the development server. There may be a way to publish multiple endpoints, but this needs further investigation.
- If the log consumer device cannot be started due to an OmniORB exception saying that the end point cannot be created, it is possible that the `142.73.34.173` needs to change to something else. It is not yet clear why this can happen. To change it do the following:
  - Remove the ORB arguments from the `talondx-log-consumer` make target, and then start the log consumer.
  - Open up Jive and look at what IP address is automatically assigned to the log consumer device. This is the IP address that we now need to use for the endpoint.
  - Find the iptables rule that maps `169.254.100.88:60721` to `142.73.34.173:60721`, and change it to the new IP address.
  - Add the ORB arguments back in, using the correct IP address for the end point.

CHAPTER

SIX

CODE

6.1 CbfComponentManager

6.1.1 CbfComponentManager Class

class ska_mid_cbf_mcs.component.component_manager.**CbfComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: BaseComponentManager

A base component manager for SKA Mid.CBF MCS

This class exists to modify the interface of the ska_tango_base.base.component_manager.BaseComponentManager. The BaseComponentManager accepts an op_state_model argument, and is expected to interact directly with it. This is not a very good design decision. It is better to leave the op_state_model behind in the device, and drive it indirectly through callbacks.

Therefore this class accepts three callback arguments: one for when communication with the component changes, one for when the power mode of the component changes, and one for when the component fault status changes. In the last two cases, callback hooks are provided so that the component can indicate the change to this component manager.

start_communicating() → None

Start communicating with the component.

stop_communicating() → None

Break off communicating with the component.

update_communication_status(*communication_status: CommunicationStatus*) → None

Handle a change in communication status.

This is a helper method for use by subclasses.

Parameters

communication_status – the new communication status of the component manager.

property is_communicating: bool

Return communication with the component is established.

SKA Mid.CBF MCS uses the more expressive *communication_status* for this, but this is still needed as a base classes hook.

Returns

whether communication with the component is established.

property communication_status: CommunicationStatus

Return the communication status of this component manager.

This is implemented as a replacement for the `is_communicating` property, which should be deprecated.

Returns
status of the communication channel with the component.

update_component_power_mode(*power_mode: Optional[ska_tango_base.control_model.PowerMode]*) → None

Update the power mode, calling callbacks as required.

This is a helper method for use by subclasses.

Parameters
power_mode – the new power mode of the component. This can be None, in which case the internal value is updated but no callback is called. This is useful to ensure that the callback is called next time a real value is pushed.

component_power_mode_changed(*power_mode: ska_tango_base.control_model.PowerMode*) → None

Handle notification that the component's power mode has changed.

This is a callback hook, to be passed to the managed component.

Parameters
power_mode – the new power mode of the component

property power_mode: Optional[ska_tango_base.control_model.PowerMode]

Return the power mode of this component manager.

Returns
the power mode of this component manager.

update_component_fault(*faulty: Optional[bool]*) → None

Update the component fault status, calling callbacks as required.

This is a helper method for use by subclasses.

Parameters
faulty – whether the component has faulted. If `False`, then this is a notification that the component has *recovered* from a fault.

component_fault_changed(*faulty: bool*) → None

Handle notification that the component's fault status has changed.

This is a callback hook, to be passed to the managed component.

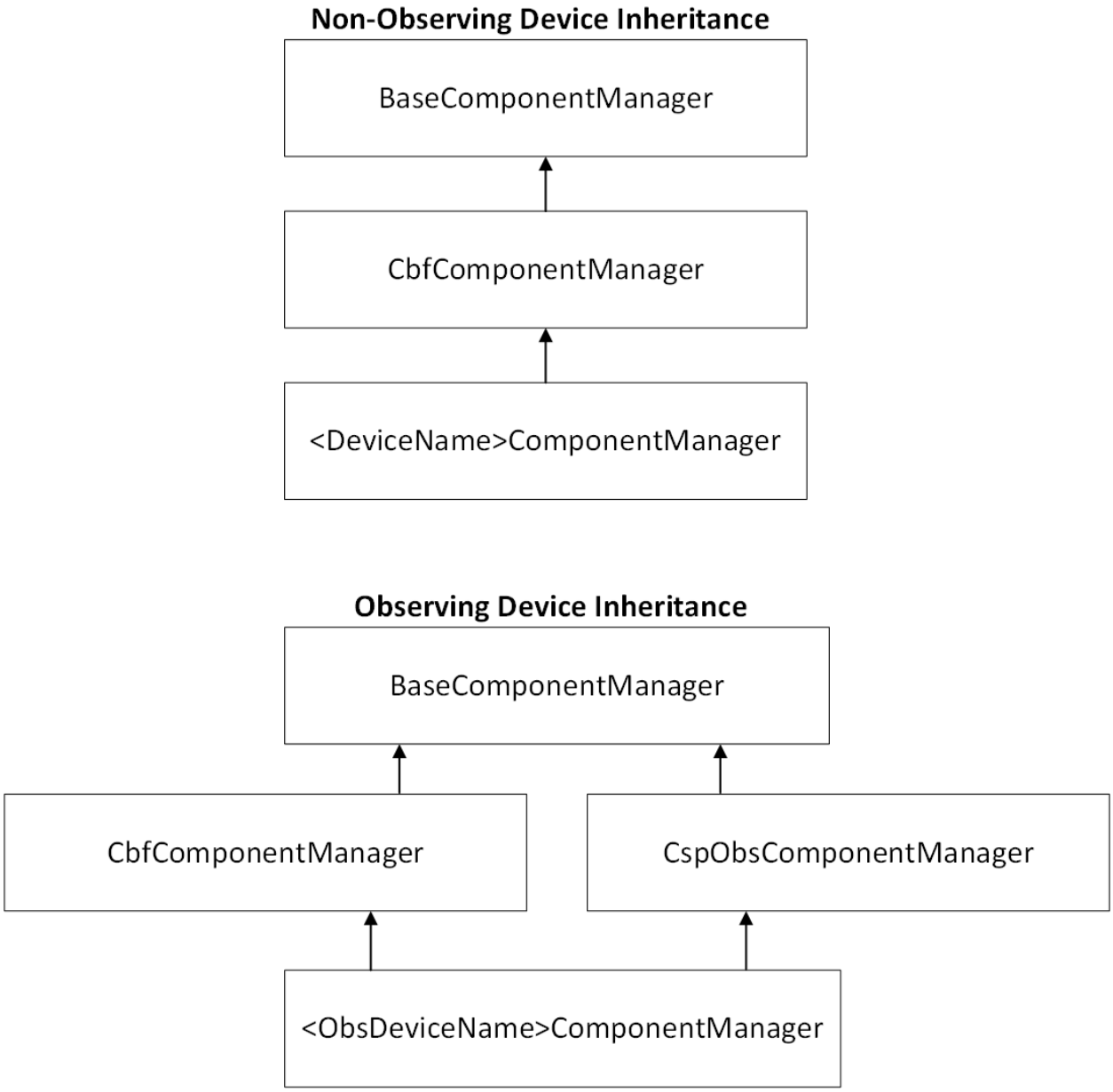
Parameters
faulty – whether the component has faulted. If `False`, then this is a notification that the component has *recovered* from a fault.

property faulty: Optional[bool]

Return whether this component manager is currently experiencing a fault.

Returns
whether this component manager is currently experiencing a fault.

The MCS contains two types of Tango devices: observing and non-observing. Non-observing devices contain only an `op_state_model` while observing devices contain both an `op_state_model` and `obs_state_model`. As shown in the inheritance diagram below, non-observing devices inherit from `CbfComponentManager` while observing devices inherit from `CbfComponentManager` and `CspObsComponentManager`.


Non-Observing Device Inheritance
BaseComponentManager
CbfComponentManager
<DeviceName>ComponentManager
Observing Device Inheritance
BaseComponentManager
CbfComponentManager
CspObsComponentManager
<ObsDeviceName>ComponentManager

6.2 CbfController

6.2.1 CbfController Class

class ska_mid_cbf_mcs.controller.controller_device.**CbfController**(**args: Any*, ***kwargs: Any*)

Bases: SKAController

CbfController TANGO device class. Primary point of contact for monitoring and control of Mid.CBF. Implements state and mode indicators, and a set of state transition commmands.

init_command_objects() → None

Sets up the command objects

get_num_capabilities() → None

Get number of capabilities for _init_Device. If property not found in db, then assign a default amount(197,27,16)

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

Bases: InitCommand

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation. :return: A tuple containing a return code and a string message indicating status. The message is for information purpose only. :return: (ResultCode, str)

always_executed_hook() → None

Hook to be executed before any command.

create_component_manager() → *ControllerComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device.

delete_device() → None

Unsubscribe to events, turn all the subarrays, VCCs and FSPs off

read_commandProgress() → int

Return commandProgress attribute: percentage progress implemented for commands that result in state/mode transitions for a large number of components and/or are executed in stages (e.g power up, power down)

read_receptorToVcc() → List[str]

Return 'receptorID:vccID'

read_vccToReceptor() → List[str]

Return receptorToVcc attribute: 'vccID:receptorID'

read_subarrayconfigID() → List[str]

Return subarrayconfigID atrribute: ID of subarray config. Used for debug purposes. empty string if subarray is not configured for a scan

read_frequencyOffsetK() → List[int]

Return frequencyOffsetK attribute: array of integers reporting receptors in subarray

write_frequencyOffsetK(*value: List[int]*) → None

Set frequencyOffsetK attribute

read_frequencyOffsetDeltaF() → List[int]

Return frequencyOffsetDeltaF attribute: Frequency offset (delta f) of all 197 receptors as an array of ints.

write_frequencyOffsetDeltaF(*value: List[int]*) → None

Set the frequencyOffsetDeltaF attribute

write_simulationMode(*value: ska_tango_base.control_model.SimulationMode*) → None

Set the Simulation Mode of the device.

Parameters
value – SimulationMode

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

Bases: OnCommand

A class for the CbfController's On() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for On() command functionality.

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

Return type
(ResultCode, str)

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

Bases: OffCommand

A class for the CbfController's Off() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Off() command functionality.

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

Return type
(ResultCode, str)

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

Bases: StandbyCommand

A class for the CbfController's Standby() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Standby() command functionality. Turn off subarray, vcc, fsp, turn CbfController to standby

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

Return type
(ResultCode, str)

6.2.2 ControllerComponentManager Class

class ska_mid_cbf_mcs.controller.controller_component_manager.**ControllerComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*

A component manager for the CbfController device.

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component

on() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn on the controller and its subordinate devices

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

Return type
(ResultCode, str)

off() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn off the controller and its subordinate devices

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

Return type
(ResultCode, str)

standby() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn the controller into low power standby mode

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

Return type
(ResultCode, str)

6.2.3 TalonDxComponentManager Class

The `TalonDxComponentManager` is used to configure and start up Tango device servers on the Talon boards. These actions are performed during the On command of the `CbfController`. Note that these actions are not executed when the `simulationMode` of the `CbfController` is set to 1 (this is the default). Prior to sending the On command to the `CbfController`, the `simulationMode` should be set to 0 if it is desired to test the command with the Talon boards in the loop.

class ska_mid_cbf_mcs.controller.talondx_component_manager.**TalonDxComponentManager**(*talondx_config_path: str*, *hw_config_path: str*, *simulation_mode: ska_tango_base.control_m*, *logger: Logger*)

Bases: object

A component manager for the Talon-DX boards. Used to configure and start the Tango applications on the HPS of each board.

configure_talons() → ska_tango_base.commands.ResultCode

Performs all actions to configure the Talon boards after power on and start the HPS device servers. This includes: copying the device server binaries and FPGA bitstream to the Talon boards, starting the HPS master device server and sending the configure command to each DsHpsMaster.

Returns
ResultCode.FAILED if any operations failed, else ResultCode.OK

_setup_tango_host_file() → None

Copy the hps_master_mcs.sh file from mnt into mnt/talondx-config

Returns
ResultCode.OK if all artifacts were copied successfully, otherwise ResultCode.FAILED

_secure_copy(*ssh_client: paramiko.SSHClient*, *src: str*, *dest: str*) → None

Execute a secure file copy to the specified target address.

Parameters

- **ssh_client** – SSH client for the Talon board we are trying to SCP to
- **src** – Source file path
- **dest** – Destination file path

Raises
SCPException – if the file copy fails

_configure_talon_networking() → ska_tango_base.commands.ResultCode

Configure the networking of the boards including DNS nameserver and ifconfig for default gateway

Returns
ResultCode.OK if all artifacts were copied successfully, otherwise ResultCode.FAILED

_copy_binaries_and_bitstream() → ska_tango_base.commands.ResultCode

Copy the relevant device server binaries and FPGA bitstream to each Talon board.

Returns
ResultCode.OK if all artifacts were copied successfully, otherwise ResultCode.FAILED

_start_hps_master() → ska_tango_base.commands.ResultCode

Start the DsHpsMaster on each Talon board.

Returns
ResultCode.OK if all HPS masters were started successfully, otherwise ResultCode.FAILED

_create_hps_master_device_proxies() → ska_tango_base.commands.ResultCode

Attempt to create a device proxy to each DsHpsMaster device.

Returns
ResultCode.OK if all proxies were created successfully, otherwise ResultCode.FAILED

_configure_hps_master() → ska_tango_base.commands.ResultCode

Send the configure command to all the DsHpsMaster devices.

Returns
ResultCode.OK if all configure commands were sent successfully, otherwise ResultCode.FAILED

6.3 CbfSubarray

6.3.1 CbfSubarray Class

6.3.2 CbfSubarray Component Class

6.4 Fsp

6.4.1 Fsp Class

class ska_mid_cbf_mcs.fsp.fsp_device.**Fsp**(**args: Any*, ***kwargs: Any*)

Bases: SKACapability

Fsp TANGO device class for the prototype

init_command_objects() → None

Sets up the command objects

always_executed_hook() → None

Hook to be executed before any commands.

create_component_manager() → *FspComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device.

delete_device() → None

Hook to delete device.

write_simulationMode(*value: ska_tango_base.control_model.SimulationMode*) → None

Set the simulation mode of the device.

Parameters
value – SimulationMode

read_functionMode() → tango.DevEnum

Read the functionMode attribute.

Returns
a DevEnum representing the mode.

Return type
tango.DevEnum

read_subarrayMembership() → List[int]

Read the subarrayMembership attribute.

Returns
an array of affiliations of the FSP.

Return type
List[int]

read_scanID() → int

Read the scanID attribute.

Returns
the scanID attribute.

Return type
int

read_configID() → str

Read the configID attribute.

Returns
the configID attribute.

Return type
str

write_configID(*value: str*) → None

Write the configID attribute.

Parameters
value – the configID value.

read_jonesMatrix() → str

Read the jonesMatrix attribute.

Returns
the jonesMatrix attribute.

Return type
string

read_delayModel() → str

Read the delayModel attribute.

Returns
the delayModel attribute.

Return type
string

read_timingBeamWeights() → str

Read the timingBeamWeights attribute.

Returns
the timingBeamWeights attribute.

Return type
string

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

Bases: InitCommand

A class for the Fsp's init_device() "command".

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

Returns

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

Return type

(ResultCode, str)

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

Bases: OnCommand

A class for the Fsp's On() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for On() command functionality.

Returns

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

Return type

(ResultCode, str)

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

Bases: OffCommand

A class for the Fsp's Off() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Off() command functionality.

Returns

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

Return type

(ResultCode, str)

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

Bases: StandbyCommand

A class for the Fsp's Standby() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Standby() command functionality.

Returns

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

Return type

(ResultCode, str)

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

Bases: ResponseCommand

A class for the Fsp's SetFunctionMode() command.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for SetFunctionMode() command functionality.

Parameters
argin – one of 'IDLE','CORR','PSS-BF','PST-BF', or 'VLBI'

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

Return type
(ResultCode, str)

SetFunctionMode(*argin: str*) → None

Set the Fsp Function Mode, either IDLE, CORR, PSS-BF, PST-BF, or VLBI If IDLE set the pss, pst, corr and vlbi devicess to DISABLE. OTherwise, turn one of them ON according to argin, and all others DISABLE.

Parameters
argin – one of 'IDLE','CORR','PSS-BF','PST-BF', or 'VLBI'

is_SetFunctionMode_allowed() → bool

Determine if SetFunctionMode is allowed (allowed if FSP state is ON).

Returns
if SetFunctionMode is allowed

Return type
bool

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

Bases: ResponseCommand

A class for the Fsp's AddSubarrayMembership() command.

do(*argin: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for AddSubarrayMembership() command functionality.

Parameters
argin – an integer representing the subarray affiliation

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

Return type
(ResultCode, str)

AddSubarrayMembership(*argin: str*) → None

Add a subarray to the subarrayMembership list.

Parameters
argin – an integer representing the subarray affiliation

is_AddSubarrayMembership_allowed() → bool

Determine if AddSubarrayMembership is allowed (allowed if FSP state is ON).

Returns
if AddSubarrayMembership is allowed

Return type
bool

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

Bases: ResponseCommand

A class for the Fsp's RemoveSubarrayMembership() command.

do(argin: int) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for RemoveSubarrayMembership() command functionality.

Parameters
argin – an integer representing the subarray affiliation

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

Return type
(ResultCode, str)

RemoveSubarrayMembership(argin: str) → None

Remove subarray from the subarrayMembership list. If subarrayMembership is empty after removing (no subarray is using this FSP), set function mode to empty.

Parameters
argin – an integer representing the subarray affiliation

is_RemoveSubarrayMembership_allowed() → bool

Determine if RemoveSubarrayMembership is allowed (allowed if FSP state is ON).

Returns
if RemoveSubarrayMembership is allowed

Return type
bool

getConfigID() → str

Get the configID for all the fspCorrSubarray

Returns
the configID

Return type
str

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

Bases: ResponseCommand

A class for the Fsp's UpdateJonesMatrix() command.

do(argin: str) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for UpdateJonesMatrix() command functionality.

Parameters
argin – the jones matrix data

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

Return type
(ResultCode, str)

UpdateJonesMatrix(argin: str) → None

Update the FSP's jones matrix (serialized JSON object)

Parameters
argin – the jones matrix data

`is_UpdateJonesMatrix_allowed`() → bool

Determine if UpdateJonesMatrix is allowed (allowed if FSP state is ON and ObsState is READY OR SCANNINNG).

Returns
if UpdateJonesMatrix is allowed

Return type
bool

`class UpdateDelayModelCommand`(**args: Any*, ***kwargs: Any*)

Bases: `ResponseCommand`

A class for the Fsp's UpdateDelayModel() command.

`do`(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for UpdateDelayModel() command functionality.

Parameters
argin – the delay model data

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

Return type
(ResultCode, str)

`UpdateDelayModel`(*argin: str*) → None

Update the FSP's delay model (serialized JSON object)

Parameters
argin – the delay model data

`is_UpdateDelayModel_allowed`() → bool

Determine if UpdateDelayModelis allowed (allowed if FSP state is ON and ObsState is READY OR SCANNINNG).

Returns
if UpdateDelayModel is allowed

Return type
bool

`class UpdateTimingBeamWeightsCommand`(**args: Any*, ***kwargs: Any*)

Bases: `ResponseCommand`

A class for the Fsp's UpdateTimingBeamWeights() command.

`do`(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for UpdateTimingBeamWeights() command functionality.

Parameters
argin – the timing beam weight data

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

Return type
(ResultCode, str)

`UpdateTimingBeamWeights`(*argin: str*) → None

Update the FSP's timing beam weights (serialized JSON object)

Parameters
argin – the timing beam weight data

is_UpdateTimingBeamWeights_allowed() → bool

Determine if UpdateTimingBeamWeights is allowed (allowed if FSP state is ON and ObsState is READY OR SCANNINNG).

Returns
if UpdateTimingBeamWeights is allowed

Return type
bool

6.4.2 FspComponentManager Class

class ska_mid_cbf_mcs.fsp.fsp_component_manager.**FspComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*

A component manager for the Fsp device.

property subarray_membership: List[int]

Subarray Membership

Returns
an array of affiliations of the FSP.

Return type
List[int]

property function_mode: tango.DevEnum

Function Mode

Returns
the Fsp function mode

Return type
tango.DevEnum

property jones_matrix: str

Jones Matrix

Returns
the jones matrix

Return type
str

property delay_model: str

Delay Model

Returns
the delay model

Return type
str

property timing_beam_weights: str

Timing Beam Weights

Returns
the timing beam weights

Return type
str

property simulation_mode: ska_tango_base.control_model.SimulationMode

Get the simulation mode of the component manager.

Returns
simulation mode of the component manager

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component

remove_subarray_membership(*argin: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Remove subarray from the subarrayMembership list. If subarrayMembership is empty after removing (no subarray is using this FSP), set function mode to empty.

Parameters
argin – an integer representing the subarray affiliation

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

Return type
(ResultCode, str)

add_subarray_membership(*argin: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Add a subarray to the subarrayMembership list.

Parameters
argin – an integer representing the subarray affiliation

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

Return type
(ResultCode, str)

on() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn on the fsp and its subordinate devices

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

Return type
(ResultCode, str)

off() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn off the fsp and its subordinate devices

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

Return type
(ResultCode, str)

standby() → Tuple[ska_tango_base.commands.ResultCode, str]

Put the fsp into low power standby mode

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

Return type
(ResultCode, str)

set_function_mode(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Put the fsp into low power standby mode

Parameters
argin – one of 'IDLE','CORR','PSS-BF','PST-BF', or 'VLBI'

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

Return type
(ResultCode, str)

update_jones_matrix(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Update the FSP's jones matrix (serialized JSON object)

Parameters
argin – the jones matrix data

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

Return type
(ResultCode, str)

update_delay_model(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Update the FSP's delay model (serialized JSON object)

Parameters
argin – the delay model data

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

Return type
(ResultCode, str)

update_timing_beam_weights(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Update the FSP's timing beam weights (serialized JSON object)

Parameters
argin – the timing beam weight data

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

Return type
(ResultCode, str)

get_fsp_corr_config_id() → str

Get the configID for all the fspCorrSubarray

Returns
the configID

Return type
str

6.4.3 FspCorrSubarray Class

class ska_mid_cbf_mcs.fsp.fsp_corr_subarray_device.**FspCorrSubarray**(**args: Any*, ***kwargs: Any*)

Bases: CspSubElementObsDevice

FspCorrSubarray TANGO device class for the FspCorrSubarray prototype

init_command_objects() → None

Sets up the command objects

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

Bases: InitCommand

A class for the FspCorrSubarray's init_device() "command".

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

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

Return type
(ResultCode, str)

always_executed_hook() → None

Hook to be executed before any commands.

create_component_manager() → *FspCorrSubarrayComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device.

delete_device() → None

Hook to delete device.

read_receptors() → List[int]

Read the receptors attribute.

Returns
the list of receptors

Return type
List[int]

read_frequencyBand() → tango.DevEnum

Read the frequencyBand attribute.

Returns
the frequency band

Return type
tango.DevEnum

read_band5Tuning() → List[float]

Read the band5Tuning attribute.

Returns
the band5Tuning attribute (array of float, first element corresponds to the first stream, second to the second stream).

Return type
List[float]

read_frequencyBandOffsetStream1() → int

Read the frequencyBandOffsetStream1 attribute.

Returns
the frequencyBandOffsetStream1 attribute

Return type
int

read_frequencyBandOffsetStream2() → int

Read the frequencyBandOffsetStream2 attribute.

Returns
the frequencyBandOffsetStream2 attribute.

Return type
int

read_frequencySliceID() → int

Read the frequencySliceID attribute.

Returns
the frequencySliceID attribute.

Return type
int

read_corrBandwidth() → int

Read the corrBandwidth attribute.

Returns
the corrBandwidth attribute (bandwidth to be correlated is <Full Bandwidth>/2^bandwidth).

Return type
int

read_zoomWindowTuning() → int

Read the zoomWindowTuning attribute.

Returns
the zoomWindowTuning attribute

Return type
int

read_integrationFactor() → int

Read the integrationFactor attribute.

Returns
the integrationFactor attribute (millisecond).

Return type
int

read_channelAveragingMap() → List[List[int]]

Read the channelAveragingMap attribute.

Returns
the channelAveragingMap attribute. Consists of 2*20 array of integers(20 tupples representing 20* 744 channels). The first element is the ID of the first channel in a channel group. The second element is the averaging factor

Return type
List[List[int]]

read_visDestinationAddress() → str

Read the visDestinationAddress attribute.

Returns
the visDestinationAddress attribute. (JSON object containing info about current SDP destination addresses being used).

Return type
str

write_visDestinationAddress(*value: str*) → None

Write the visDestinationAddress attribute.

Parameters
value – the visDestinationAddress attribute value. (JSON object containing info about current SDP destination addresses being used).

read_fspChannelOffset() → int

Read the fspChannelOffset attribute.

Returns
the fspChannelOffset attribute.

Return type
int

write_fspChannelOffset(*value: int*) → None

Write the fspChannelOffset attribute.

Parameters
value – the fspChannelOffset attribute value.

read_outputLinkMap() → List[List[int]]

Read the outputLinkMap attribute.

Returns
the outputLinkMap attribute.

Return type
List[List[int]]

write_outputLinkMap(*value: List[List[int]]*) → None
Write the outputLinkMap attribute.

Parameters
value – the outputLinkMap attribute value.

read_scanID() → int
Read the scanID attribute.

Returns
the scanID attribute.

Return type
int

write_scanID(*value: int*) → None
Write the scanID attribute.

Parameters
value – the scanID attribute value.

read_configID() → str
Read the configID attribute.

Returns
the configID attribute.

Return type
str

write_configID(*value: str*) → None
Write the configID attribute.

Parameters
value – the configID attribute value.

write_simulationMode(*value: ska_tango_base.control_model.SimulationMode*) → None
Set the simulation mode of the device.

Parameters
value – SimulationMode

class OnCommand(**args: Any*, ***kwargs: Any*)
Bases: OnCommand

A class for the FspCorrSubarray's On() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]
Stateless hook for On() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

class OffCommand(**args: Any*, ***kwargs: Any*)
Bases: OffCommand

A class for the FspCorrSubarray's Off() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Off() command functionality.

Returns

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

Return type

(ResultCode, str)

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

Bases: StandbyCommand

A class for the FspCorrSubarray's Standby() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Standby() command functionality.

Returns

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

Return type

(ResultCode, str)

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

Bases: ConfigureScanCommand

A class for the FspCorrSubarray's ConfigureScan() command.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for ConfigureScan() command functionality.

Parameters

argin (*str*) – The configuration as JSON formatted string

Returns

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

Return type

(ResultCode, str)

Raises

CommandError if the configuration data validation fails.

validate_input(*argin: str*) → Tuple[bool, str]

Validate the configuration parameters against allowed values, as needed.

Parameters

argin – The JSON formatted string with configuration for the device. :type argin: 'DevString'

Returns

A tuple containing a boolean and a string message.

Return type

(bool, str)

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

Bases: ScanCommand

A class for the FspCorrSubarrFspCorrSubarrayay's Scan() command.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.

Parameters

argin (*str*) – The scan ID

Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)
Raises
CommandError if the configuration data validation fails.

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

Bases: EndScanCommand

A class for the FspCorrSubarray's Scan() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)
Raises
CommandError if the configuration data validation fails.

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

Bases: GoToIdleCommand

A class for the FspCorrSubarray's GoToIdle command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for GoToIdle() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

is_getLinkAndAddress_allowed() → bool

Determine if getLinkAndAddress is allowed (allowed if destination addresses are received, meaning outputLinkMap also received (checked in subarray validate scan)).

Returns
if getLinkAndAddress is allowed

Return type
bool

getLinkAndAddress(*argin: int*) → str

Get output link and destination addresses in JSON based on a channel ID.

Parameters
argin – the channel id.

Returns
the output link and destination addresses in JSON.

Return type
str

6.4.4 FspCorrSubarrayComponentManager Class

class ska_mid_cbf_mcs.fsp.fsp_corr_subarray_component_manager.**FspCorrSubarrayComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*, CspObsComponentManager

A component manager for the FspCorrSubarray device.

property frequency_band: tango.DevEnum

Frequency Band

Returns
the frequency band

Return type
tango.DevEnum

property stream_tuning: List[float]

Band 5 Tuning

Returns
an array of float, (first element corresponds to the first stream, second to the second stream).

Return type
List[float]

property frequency_band_offset_stream1: int

Frequency Band Offset Stream 1

Returns
the frequency band offset for stream 1

Return type
int

property frequency_band_offset_stream2: int

Frequency Band Offset Stream 2

Returns
the frequency band offset for stream 2

Return type
int

property frequency_slice_id: int

Frequency Slice ID

Returns
the frequency slice id

Return type
int

property bandwidth: int

Bandwidth

Returns
the corr bandwidth (bandwidth to be correlated is <Full Bandwidth>/2^bandwidth).

Return type
int

property integration_factor: int

Integration Factor

Returns
the integration factor

Return type
int

property fsp_channel_offset: int

FSP Channel Offset

Returns
the FSP channel offset

Return type
int

property vis_destination_address: str

VIS Destination Address

Returns
JSON string containing info about current SDP destination addresses being used

Return type
str

property output_link_map: List[List[int]]

Output Link Map

Returns
the output link map

Return type
List[List[int]]

property channel_averaging_map: List[List[int]]

Channel Averaging Map

Returns
the channel averaging map. Consists of 2*20 array of integers(20 tupples representing 20* 744 channels). The first element is the ID of the first channel in a channel group. The second element is the averaging factor

Return type
List[List[int]]

property zoom_window_tuning: int

Zoom Window Tuning

Returns
the zoom window tuning

Return type
int

property config_id: str

Config ID

Returns
the config id

Return type
str

property scan_id: int

Scan ID

Returns
the scan id

Return type
int

property receptors: List[int]

Receptors

Returns
list of receptor ids

Return type
List[int]

property simulation_mode: ska_tango_base.control_model.SimulationMode

Get the simulation mode of the component manager.

Returns
simulation mode of the component manager

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component

configure_scan(*configuration: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the ConfigureScan() command functionality

Parameters
configuration – The configuration as JSON formatted string

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

Return type
(ResultCode, str)

scan(*scan_id: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the Scan() command functionality

Parameters
scan_id – The scan id

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

Return type
(ResultCode, str)

end_scan() → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the EndScan() command functionality

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

Return type
(ResultCode, str)

go_to_idle() → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the GoToIdle() command functionality

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

Return type
(ResultCode, str)

6.4.5 FspPssSubarray Class

class ska_mid_cbf_mcs.fsp.fsp_pss_subarray_device.**FspPssSubarray**(**args: Any*, ***kwargs: Any*)

Bases: CspSubElementObsDevice

FspPssSubarray TANGO device class for the FspPssSubarray prototype

init_command_objects() → None

Sets up the command objects

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

Bases: InitCommand

A class for the FspPssSubarray's init_device() "command".

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

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

Return type
(ResultCode, str)

always_executed_hook() → None

Hook to be executed before any commands.

create_component_manager() → *FspPssSubarrayComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device.

delete_device() → None

Hook to delete device.

read_receptors() → List[int]

Read the receptors attribute.

Returns
the receptors attribute.

Return type
List[int]

`read_searchBeams()` → List[str]

Read the searchBeams attribute.

Returns
the searchBeams attribute.

Return type
List[str]

`read_searchBeamID()` → List[int]

Read the searchBeamID attribute.

Returns
the searchBeamID attribute.

Return type
List[int]

`read_searchWindowID()` → List[int]

Read the searchWindowID attribute.

Returns
the searchWindowID attribute.

Return type
List[int]

`read_outputEnable()` → bool

Read the outputEnable attribute. Used to enable/disable transmission of the output products.

Returns
the outputEnable attribute.

Return type
bool

`read_scanID()` → int

Read the scanID attribute.

Returns
the scanID attribute.

Return type
int

`write_scanID`(*value: int*) → None

Write the scanID attribute.

Parameters
value – the scanID attribute value.

`read_configID()` → str

Read the configID attribute.

Returns
the configID attribute.

Return type
str

`write_configID`(*value: str*) → None

Write the configID attribute.

Parameters
value – the configID attribute value.

`class OnCommand`(**args: Any*, ***kwargs: Any*)

Bases: `OnCommand`

A class for the FspPssSubarray's On() command.

`do`() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for On() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

`class OffCommand`(**args: Any*, ***kwargs: Any*)

Bases: `OffCommand`

A class for the FspPssSubarray's Off() command.

`do`() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Off() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

`class StandbyCommand`(**args: Any*, ***kwargs: Any*)

Bases: `StandbyCommand`

A class for the FspPssSubarray's Standby() command.

`do`() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Standby() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

`class ConfigureScanCommand`(**args: Any*, ***kwargs: Any*)

Bases: `ConfigureScanCommand`

A class for the FspPssSubarray's ConfigureScan() command.

`do`(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for ConfigureScan() command functionality.
Parameters
argin (`str`) – The configuration as JSON formatted string.

Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)
Raises
CommandError if the configuration data validation fails.

validate_input(*argin: str*) → Tuple[bool, str]

Validate the configuration parameters against allowed values, as needed.
Parameters
argin – The JSON formatted string with configuration for the device. :type argin: 'DevString'
Returns
A tuple containing a boolean and a string message.
Return type
(bool, str)

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

Bases: ScanCommand

A class for the FspPssSubarray's Scan() command.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.
Parameters
argin (*str*) – The scan ID
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)
Raises
CommandError if the configuration data validation fails.

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

Bases: EndScanCommand

A class for the FspPssSubarray's Scan() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)
Raises
CommandError if the configuration data validation fails.

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

Bases: GoToIdleCommand

A class for the FspPssSubarray's GoToIdle command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for GoToIdle() command functionality.

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

Return type
(ResultCode, str)

6.4.6 FspPssSubarrayComponentManager Class

class ska_mid_cbf_mcs.fsp.fsp_pss_subarray_component_manager.**FspPssSubarrayComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*, CspObsComponentManager

A component manager for the FspPssSubarray device.

property scan_id: int
Scan ID

Returns
the scan id

Return type
int

property config_id: str
Config ID

Returns
the config id

Return type
str

property fsp_id: int
Fsp ID

Returns
the fsp id

Return type
int

property search_window_id: int
Search Window ID

Returns
the search window id

Return type
int

property search_beams: List[str]
Search Beams

Returns
search beams

Return type
List[str]

property search_beam_id: List[int]

Search Beam ID

Returns
search beam id

Return type
List[int]

property output_enable: bool

Output Enable

Returns
output enable

Return type
bool

property receptors: List[int]

Receptors

Returns
list of receptor ids

Return type
List[int]

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component

configure_scan(*configuration: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the ConfigureScan() command functionality

Parameters
configuration – The configuration as JSON formatted string

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

Return type
(ResultCode, str)

scan(*scan_id: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the Scan() command functionality

Parameters
scan_id – The scan id

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

Return type
(ResultCode, str)

end_scan() → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the EndScan() command functionality

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

Return type
(ResultCode, str)

`go_to_idle()` → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the GoToIdle() command functionality

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

Return type
(ResultCode, str)

6.4.7 FspPstSubarray Class

`class ska_mid_cbf_mcs.fsp.fsp_pst_subarray_device.FspPstSubarray(*args: Any, **kwargs: Any)`

Bases: `CspSubElementObsDevice`

FspPstSubarray TANGO device class for the FspPstSubarray prototype

`init_command_objects()` → None

Sets up the command objects

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

Bases: `InitCommand`

A class for the FspPstSubarray's init_device() "command".

`do()` → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

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

Return type
(ResultCode, str)

`always_executed_hook()` → None

Hook to be executed before any commands.

`create_component_manager()` → *FspPstSubarrayComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device.

`delete_device()` → None

Hook to delete device.

`read_outputEnable()` → bool

Read the outputEnable attribute. Used to enable/disable transmission of the output products.

Returns
the outputEnable attribute.

Return type
bool

read_receptors() → List[int]

Read the receptors attribute.

Returns
the list of receptors.

Return type
List[int]

write_receptors(*value: List[int]*) → None

Write the receptors attribute.

Parameters
value – the receptors attribute value.

read_timingBeams() → List[str]

Read the timingBeams attribute.

Returns
the timingBeams attribute.

Return type
List[int]

read_timingBeamID() → List[int]

Read the list of Timing Beam IDs.

Returns
the timingBeamID attribute.

Return type
List[int]

read_scanID() → int

Read the scanID attribute.

Returns
the scanID attribute.

Return type
int

write_scanID(*value: int*) → None

Write the scanID attribute.

Parameters
value – the scanID attribute value.

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

Bases: OnCommand

A class for the FspPstSubarray's On() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for On() command functionality.

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

Return type
(ResultCode, str)

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

Bases: OffCommand

A class for the FspPstSubarray's Off() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Off() command functionality.

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

Return type
(ResultCode, str)

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

Bases: StandbyCommand

A class for the FspPstSubarray's Standby() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Standby() command functionality.

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

Return type
(ResultCode, str)

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

Bases: ConfigureScanCommand

A class for the FspPstSubarray's ConfigureScan() command.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for ConfigureScan() command functionality.

Parameters
argin (*str*) – The configuration as JSON formatted string

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

Return type
(ResultCode, str)

Raises
CommandError if the configuration data validation fails.

validate_input(*argin: str*) → Tuple[bool, str]

Validate the configuration parameters against allowed values, as needed.

Parameters
argin – The JSON formatted string with configuration for the device. :type argin: 'DevString'

Returns
A tuple containing a boolean and a string message.

Return type
(bool, str)

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

Bases: ScanCommand

A class for the FspPstSubarray's Scan() command.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.

Parameters
argin (*str*) – The scan ID

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

Return type
(ResultCode, str)

Raises
CommandError if the configuration data validation fails.

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

Bases: EndScanCommand

A class for the FspPstSubarray's EndScan() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.

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

Return type
(ResultCode, str)

Raises
CommandError if the configuration data validation fails.

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

Bases: GoToIdleCommand

A class for the FspPstSubarray's GoToIdle command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for GoToIdle() command functionality.

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

Return type
(ResultCode, str)

6.4.8 FspPstSubarrayComponentManager Class

class ska_mid_cbf_mcs.fsp.fsp_pst_subarray_component_manager.**FspPstSubarrayComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*, CspObsComponentManager

A component manager for the FspPstSubarray device.

property fsp_id: int

Fsp ID

Returns
the fsp id

Return type
int

property timing_beams: List[str]

Timing Beams

Returns
the timing beams

Return type
List[str]

property timing_beam_id: List[int]

Timing Beam ID

Returns
list of timing beam ids

Return type
List[int]

property receptors: List[int]

Receptors

Returns
list of receptor ids

Return type
List[int]

property scan_id: int

Scan ID

Returns
the scan id

Return type
int

property output_enable: bool

Output Enable

Returns
output enable

Return type
bool

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component

configure_scan(*configuration: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the ConfigureScan() command functionality

Parameters
configuration – The configuration as JSON formatted string

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

Return type
(ResultCode, str)

scan(*scan_id: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the Scan() command functionality

Parameters
scan_id – The scan id

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

Return type
(ResultCode, str)

end_scan() → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the EndScan() command functionality

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

Return type
(ResultCode, str)

go_to_idle() → Tuple[ska_tango_base.commands.ResultCode, str]

Performs the GoToIdle() command functionality

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

Return type
(ResultCode, str)

6.5 Vcc

6.5.1 VCC Class

class ska_mid_cbf_mcs.vcc.vcc_device.**Vcc**(**args: Any*, ***kwargs: Any*)

Bases: CspSubElementObsDevice

Vcc TANGO device class for the prototype

init_command_objects() → None

Sets up the command objects

create_component_manager() → *VccComponentManager*

always_executed_hook() → None

Hook to be executed before any commands.

delete_device() → None

Hook to delete device.

write_simulationMode(*value: ska_tango_base.control_model.SimulationMode*) → None

Set the simulation mode of the device.

Parameters
value – SimulationMode

read_receptorID() → int

Read the receptorID attribute.

Returns
the Vcc's receptor id.

Return type
int

write_receptorID(*value: int*) → None

Write the receptorID attribute.

Parameters
value – the receptorID value.

read_subarrayMembership() → int

Read the subarrayMembership attribute.

Returns
the subarray membership (0 = no affiliation).

Return type
int

write_subarrayMembership(*value: int*) → None

Write the subarrayMembership attribute.

Parameters
value – the subarray membership value (0 = no affiliation).

read_frequencyOffsetK() → int

Read the frequencyOffsetK attribute.

Returns
the frequency offset k-value

Return type
int

write_frequencyOffsetK(*value: int*) → None

Write the frequencyOffsetK attribute.

Parameters
value – the frequency offset k-value

read_frequencyOffsetDeltaF() → int

Read the frequencyOffsetDeltaF attribute.

Returns
the frequency offset delta-f value

Return type
int

write_frequencyOffsetDeltaF(*value: int*) → None

Write the frequencyOffsetDeltaF attribute.

Parameters
value – the frequency offset delta-f value

read_frequencyBand() → tango.DevEnum

Read the frequencyBand attribute.

Returns
the frequency band (being observed by the current scan, one of ["1", "2", "3", "4", "5a", "5b"]).

Return type
tango.DevEnum

read_band5Tuning() → List[float]

Read the band5Tuning attribute.

Returns
the band5Tuning attribute (stream tuning (GHz)).

Return type
list of float

read_frequencyBandOffsetStream1() → int

Read the frequencyBandOffsetStream1 attribute.

Returns
the frequencyBandOffsetStream1 attribute.

Return type
int

read_frequencyBandOffsetStream2() → int

Read the frequencyBandOffsetStream2 attribute.

Returns
the frequencyBandOffsetStream2 attribute.

Return type
int

read_dopplerPhaseCorrection() → List[float]

Read the dopplerPhaseCorrection attribute.

Returns
the dopplerPhaseCorrection attribute.

Return type
list of float

write_dopplerPhaseCorrection(*value: List[float]*) → None

Write the dopplerPhaseCorrection attribute.

Parameters
value – the dopplerPhaseCorrection attribute value.

read_rfiFlaggingMask() → str

Read the rfiFlaggingMask attribute.

Returns
the rfiFlaggingMask attribute.

Return type
str/JSON

read_delayModel() → str

Read the delayModel attribute.

Returns
the delayModel attribute (delay model coefficients, :return: the delayModel attribute (delay model coefficients,

Returns
the delayModel attribute (delay model coefficients, given per frequency slice).

Return type
list of list of float

read_jonesMatrix() → str

Read the jonesMatrix attribute.

Returns
the jonesMatrix attribute (jones matrix values, given per frequency slice).

Return type
str

read_scanID() → int

Read the scanID attribute.

Returns
the scanID attribute.

Return type
int

read_configID() → str

Read the configID attribute.

Returns
the configID attribute.

Return type
str

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

Bases: InitCommand

A class for the Vcc's init_device() "command".

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

Returns

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

Return type

(ResultCode, str)

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

Bases: OnCommand

A class for the Vcc's on command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

Returns

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

Return type

(ResultCode, str)

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

Bases: OffCommand

A class for the Vcc's off command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

Returns

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

Return type

(ResultCode, str)

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

Bases: StandbyCommand

A class for the Vcc's standby command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

Returns

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

Return type

(ResultCode, str)

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

Bases: ResponseCommand

A class for the Vcc's ConfigureBand() command.

Turn on the corresponding band device and disable all the others.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for ConfigureBand() command functionality.

Parameters

freq_band_name – the frequency band name

Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

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

Bases: ConfigureScanCommand

A class for the Vcc's ConfigureScan() command.

do(argin: str) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for ConfigureScan() command functionality.
Parameters
argin (str) – The configuration as JSON formatted string
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)
Raises
CommandError if the configuration data validation fails.

validate_input(argin: str) → Tuple[bool, str]

Validate the configuration parameters against allowed values, as needed.
Parameters
argin – The JSON formatted string with configuration for the device. :type argin: 'DevString'
Returns
A tuple containing a boolean and a string message.
Return type
(bool, str)

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

Bases: ScanCommand

A class for the Vcc's Scan() command.

do(argin: str) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for Scan() command functionality.
Parameters
argin (str) – The scan ID as JSON formatted string
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.
Return type
(ResultCode, str)

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

Bases: EndScanCommand

A class for the Vcc's EndScan() command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for EndScan() command functionality.
Returns
A tuple containing a return code and a string message indicating status. The message is for information purpose only.

Return type
(ResultCode, str)

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

Bases: ObsResetCommand

A class for the VCC's ObsReset command.

do()

Stateless hook for ObsReset() command functionality.

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

Return type
(ResultCode, str)

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

Bases: AbortCommand

A class for the VCC's Abort command.

do()

Stateless hook for Abort() command functionality.

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

Return type
(ResultCode, str)

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

Bases: GoToIdleCommand

A class for the Vcc's GoToIdle command.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for GoToIdle() command functionality.

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

Return type
(ResultCode, str)

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

Bases: BaseCommand

A class for the Vcc's UpdateDopplerPhaseCorrection() command.

Update Vcc's doppler phase correction.

is_allowed() → bool

Determine if UpdateDopplerPhaseCorrection is allowed (allowed when Devstate is ON and ObsState is READY OR SCANNING).

Returns
if UpdateDopplerPhaseCorrection is allowed

Return type
bool

do(*argin: str*) → None

Stateless hook for UpdateDopplerPhaseCorrection() command functionality.

Parameters
argin – the doppler phase correction JSON

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

Bases: BaseCommand

A class for the Vcc's UpdateDelayModel() command.

Update Vcc's delay model.

is_allowed() → bool

Determine if UpdateDelayModel is allowed (allowed when Devstate is ON and ObsState is READY OR SCANNING).
Returns
if UpdateDelayModel is allowed
Return type
bool

do(argin: str) → None

Stateless hook for UpdateDelayModel() command functionality.
Parameters
argin – the delay model JSON

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

Bases: BaseCommand

A class for the Vcc's UpdateJonesMatrix() command.

Update Vcc's Jones matrix.

is_allowed() → bool

Determine if UpdateJonesMatrix is allowed (allowed when Devstate is ON and ObsState is READY OR SCANNING).
Returns
if UpdateJonesMatrix is allowed
Return type
bool

do(argin: str) → None

Stateless hook for UpdateJonesMatrix() command functionality.
Parameters
argin – the Jones Matrix JSON

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

Bases: ResponseCommand

A class for the Vcc's ConfigureSearchWindow() command.

Configure a search window by sending parameters from the input(JSON) to SearchWindow device. This function is called by the subarray after the configuration has already been validated.

is_allowed() → bool

Determine if ConfigureSearchWindow is allowed (allowed if DevState is ON and ObsState is CONFIGURING)
Returns
if ConfigureSearchWindow is allowed
Return type
bool

validate_input(*argin: str*) → Tuple[bool, str]

Validate a search window configuration

Parameters

argin – JSON object with the search window parameters

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for ConfigureSearchWindow() command functionality.

Parameters

argin – JSON object with the search window parameters

Returns

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

Return type

(ResultCode, str)

6.5.2 VccComponentManager Class

class ska_mid_cbf_mcs.vcc.vcc_component_manager.**VccComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*, CspObsComponentManager

Component manager for Vcc class.

property config_id: str

Configuration ID

Returns

the configuration ID

property scan_id: int

Scan ID

Returns

the scan ID

property receptor_id: int

Receptor ID

Returns

the receptor ID

property frequency_offset_k: int

Frequency Offset K-value for this receptor

Returns

the frequency offset k-value

property frequency_offset_delta_f: int

Frequency Offset Delta-F Value for this receptor

Returns

the frequency offset delta-f value

property frequency_band: int

Frequency Band

Returns
the frequency band as the integer index in an array of frequency band labels: [“1”, “2”, “3”, “4”, “5a”, “5b”]

property stream_tuning: List[float]

Band 5 Stream Tuning

Returns
the band 5 stream tuning

property frequency_band_offset_stream1: int

Frequency Band Offset Stream 1

Returns
the frequency band offset for stream 1

property frequency_band_offset_stream2: int

Frequency Band Offset Stream 2

Returns
the frequency band offset for stream 2, this is only use when band 5 is active

property rfi_flagging_mask: str

RFI Flagging Mask

Returns
the RFI flagging mask

property jones_matrix: str

Jones Matrix

Returns
the last received Jones matrix

property delay_model: str

Delay Model

Returns
the last received delay model

property doppler_phase_correction: List[float]

Doppler Phase Correction

Returns
the last received Doppler phase correction array

property simulation_mode: ska_tango_base.control_model.SimulationMode

Get the simulation mode of the component manager.

Returns
simulation mode of the component manager

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component.

on() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn on VCC component. This attempts to establish communication with the VCC devices on the HPS.

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

Return type
(ResultCode, str)

Raises
ConnectionError – if unable to connect to HPS VCC devices

off() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn off VCC component; currently unimplemented.

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

Return type
(ResultCode, str)

standby() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn VCC component to standby; currently unimplemented.

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

Return type
(ResultCode, str)

configure_band(*freq_band_name: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Configure the corresponding band. At the HPS level, this reconfigures the FPGA to the correct bitstream and enables the respective band device. All other band devices are disabled.

Parameters
freq_band_name – the frequency band name

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

Return type
(ResultCode, str)

deconfigure() → None

Deconfigure scan configuration parameters.

configure_scan(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Execute configure scan operation.

Parameters
argin – JSON string with the configure scan parameters

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

Return type
(ResultCode, str)

scan(*scan_id: int*) → Tuple[ska_tango_base.commands.ResultCode, str]

Begin scan operation.

Parameters
argin – scan ID integer

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

Return type
(ResultCode, str)

end_scan() → Tuple[ska_tango_base.commands.ResultCode, str]

End scan operation.

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

Return type
(ResultCode, str)

abort()

Tell the current VCC band device to abort whatever it was doing.

obsreset()

Reset the configuration.

configure_search_window(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Configure a search window by sending parameters from the input(JSON) to SearchWindow self. This function is called by the subarray after the configuration has already been validated, so the checks here have been removed to reduce overhead.

Parameters
argin – JSON string with the search window parameters

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

Return type
(ResultCode, str)

update_doppler_phase_correction(*argin: str*) → None

Update Vcc's doppler phase correction

Parameters
argin – the doppler phase correction JSON string

update_delay_model(*argin: str*) → None

Update Vcc's delay model

Parameters
argin – the delay model JSON string

update_jones_matrix(*argin: str*) → None

Update Vcc's jones matrix

Parameters
argin – the jones matrix JSON string

6.6 PowerSwitch

6.6.1 PowerSwitch Device

class ska_mid_cbf_mcs.power_switch.power_switch_device.**PowerSwitch**(**args: Any*, ***kwargs: Any*)

Bases: SKABaseDevice

TANGO device class for controlling and monitoring the web power switch that distributes power to the Talon LRUs.

always_executed_hook() → None

Hook to be executed before any attribute access or command.

delete_device() → None

Uninitialize the device.

create_component_manager() → *PowerSwitchComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device

init_command_objects() → None

Sets up the command objects.

write_simulationMode(*value: ska_tango_base.control_model.SimulationMode*) → None

Set the simulation mode of the device. When simulation mode is set to True, the power switch software simulator is used in place of the hardware. When simulation mode is set to False, the real power switch driver is used.

Parameters
value – SimulationMode

read_numOutlets() → int

Get the number of outlets.

Returns
number of outlets

read_isCommunicating() → bool

Get whether or not the power switch is communicating.

Returns
True if power switch can be contacted, False if not

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

Bases: InitCommand

A class for the PowerSwitch's init_device() "command".

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation.

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

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

Bases: ResponseCommand

The command class for the TurnOnOutlet command.

Turn on an individual outlet, specified by the outlet ID

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Implement TurnOnOutlet command functionality.

Parameters

argin – the outlet ID of the outlet to switch on

Returns

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

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

Bases: ResponseCommand

The command class for the TurnOffOutlet command.

Turn off an individual outlet, specified by the outlet ID.

do(*argin: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Implement TurnOffOutlet command functionality.

Parameters

argin – the outlet ID of the outlet to switch off

Returns

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

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

Bases: BaseCommand

The command class for the GetOutletPowerMode command.

Get the power mode of an individual outlet, specified by the outlet ID.

do(*argin: str*) → ska_tango_base.control_model.PowerMode

Implement GetOutletPowerMode command functionality.

Parameters

argin – the outlet ID to get the state of

Returns

power mode of the outlet

6.6.2 PowerSwitchComponentManager Class

class ska_mid_cbf_mcs.power_switch.power_switch_component_manager.**PowerSwitchComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*

A component manager for the power switch. Calls either the power switch driver or the power switch simulator based on the value of simulation mode.

Parameters

- **simulation_mode** – simulation mode identifies if the real power switch driver or the simulator should be used
- **protocol** – Connection protocol (HTTP or HTTPS) for the power switch
- **ip** – IP address of the power switch
- **login** – Login username of the power switch
- **password** – Login password for the power switch
- **logger** – a logger for this object to use

property num_outlets: int

Get number of outlets present in this power switch.

Returns
number of outlets

property is_communicating: bool

Returns whether or not the power switch can be communicated with.

Returns
whether the power switch is communicating

property simulation_mode: ska_tango_base.control_model.SimulationMode

Get the simulation mode of the component manager.

Returns
simulation mode of the component manager

start_communicating() → None

Perform any setup needed for communicating with the power switch.

stop_communicating() → None

Stop communication with the component.

get_outlet_power_mode(*outlet: str*) → ska_tango_base.control_model.PowerMode

Get the power mode of a specific outlet.

Parameters
outlet – outlet ID

Returns
power mode of the outlet

Raises
AssertionError – if outlet ID is out of bounds

turn_on_outlet(*outlet: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Tell the power switch to turn on a specific outlet.

Parameters
outlet – outlet ID to turn on

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

Raises
AssertionError – if outlet ID is out of bounds

turn_off_outlet(*outlet: str*) → Tuple[ska_tango_base.commands.ResultCode, str]

Tell the power switch to turn off a specific outlet.

Parameters
outlet – outlet ID to turn off

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

Raises
AssertionError – if outlet ID is out of bounds

get_power_switch_driver(*model: str*, *ip: str*, *login: str*, *password: str*, *logger: Logger*)

6.6.3 PowerSwitchDriver

class ska_mid_cbf_mcs.power_switch.power_switch_driver.**PowerSwitchDriver**(*protocol: str*, *ip: str*, *login: str*, *password: str*, *content_type: str*, *outlet_list_url: str*, *outlet_state_url: str*, *outlet_control_url: str*, *turn_on_action: str*, *turn_off_action: str*, *state_on: str*, *state_off: str*, *outlet_schema_file: str*, *outlet_id_list: List[str]*, *logger: Logger*)

Bases: object

A driver for the DLI web power switch.

Parameters

- **protocol** – Connection protocol (HTTP or HTTPS) for the power switch
- **ip** – IP address of the power switch
- **login** – Login username of the power switch
- **password** – Login password for the power switch
- **content_type** – The content type in the request header
- **outlet_list_url** – A portion of the URL to get the list of outlets
- **outlet_state_url** – A portion of the URL to get the outlet state
- **outlet_control_url** – A portion of the URL to turn on/off outlet
- **turn_on_action** – value to pass to request to turn on an outlet
- **turn_off_action** – value to pass to request to turn on an outlet
- **state_on** – value of the outlet's state when on
- **state_off** – value of the outlet's state when off
- **outlet_schema_file** – File name for the schema for a list of outlets

- **outlet_id_list** – List of Outlet IDs
- **logger** – a logger for this object to use

query_timeout_s = 6

Timeout in seconds used when waiting for a reply from the power switch

initialize() → None

Initializes any variables needed for further communication with the power switch. Should be called once before any of the other methods.

property num_outlets: int

Get number of outlets present in this power switch.

Returns
number of outlets

property is_communicating: bool

Returns whether or not the power switch can be communicated with.

Returns
whether the power switch is communicating

get_outlet_power_mode(*outlet: str*) → ska_tango_base.control_model.PowerMode

Get the power mode of a specific outlet.

Parameters
outlet – outlet ID

Returns
power mode of the outlet

Raises

- **AssertionError** – if outlet ID is out of bounds
- **AssertionError** – if outlet power mode is different than expected

turn_on_outlet(*outlet: str*) → tuple[ResultCode, str]

Tell the DLI power switch to turn on a specific outlet.

Parameters
outlet – outlet ID to turn on

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

Raises
AssertionError – if outlet ID is out of bounds

turn_off_outlet(*outlet: str*) → tuple[ResultCode, str]

Tell the DLI power switch to turn off a specific outlet.

Parameters
outlet – outlet ID to turn off

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

Raises
AssertionError – if outlet ID is out of bounds

get_outlet_list()

Query the power switch for a list of outlets and get their name and current state.

Returns
list of all the outlets available in this power switch, or an empty list if there was an error

6.6.4 PowerSwitchSimulator

class ska_mid_cbf_mcs.power_switch.power_switch_simulator.**PowerSwitchSimulator**(*model: str*, *logger: Logger*)

Bases: object

A simulator for the power switch.

Parameters

- **model** – Name of the power switch model
- **logger** – a logger for this object to use

property num_outlets: int

Get number of outlets present in this power switch.

Returns
number of outlets

property is_communicating: bool

Returns whether or not the power switch can be communicated with.

Returns
simulator always returns True

get_outlet_power_mode(*outlet: str*) → ska_tango_base.control_model.PowerMode

Get the power mode of a specific outlet.

Parameters
outlet – outlet ID

Returns
power mode of the outlet

Raises
AssertionError – if outlet ID is out of bounds

turn_on_outlet(*outlet: str*) → tuple[ResultCode, str]

Turn on a specific outlet.

Parameters
outlet – outlet ID to turn on

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

Raises
AssertionError – if outlet ID is out of bounds

turn_off_outlet(*outlet: str*) → tuple[ResultCode, str]

Turn off a specific outlet.

Parameters
outlet – outlet ID to turn off

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

Raises
AssertionError – if outlet ID is out of bounds

get_outlet_list()

Returns a list of 8 outlets, containing their name and current state. The current state is always set to OFF.

Returns
list of all the outlets available in this power switch

6.7 TalonLRU

6.7.1 TalonLRU Device

class ska_mid_cbf_mcs.talon_lru.talon_lru_device.**TalonLRU**(**args: Any*, ***kwargs: Any*)

Bases: SKABaseDevice

TANGO device class for controlling and monitoring a Talon LRU.

always_executed_hook() → None

Hook to be executed before any attribute access or command.

delete_device() → None

Uninitialize the device.

init_command_objects() → None

Sets up the command objects.

read_PDU1PowerMode() → ska_tango_base.control_model.PowerMode

Read the power mode of the outlet specified by PDU 1.

Returns
Power mode of PDU 1

read_PDU2PowerMode() → ska_tango_base.control_model.PowerMode

Read the power mode of the outlet specified by PDU 2.

Returns
Power mode of PDU 2

create_component_manager() → *TalonLRUComponentManager*

Create and return a component manager for this device.

Returns
a component manager for this device.

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

Bases: InitCommand

A class for the TalonLRU's init_device() "command".

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Stateless hook for device initialisation. Creates the device proxies to the power switch devices.

Returns

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

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

Bases: OnCommand

The command class for the On command.

Turn on both outlets that provide power to the LRU. Device is put into ON state if at least one outlet was successfully turned on.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Implement On command functionality.

Returns

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

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

Bases: OffCommand

The command class for the Off command.

Turn off both outlets that provide power to the LRU. Device is put in the OFF state if both outlets were successfully turned off.

do() → Tuple[ska_tango_base.commands.ResultCode, str]

Implement Off command functionality.

Returns

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

6.7.2 TalonLRUComponentManager Class

class ska_mid_cbf_mcs.talon_lru.talon_lru_component_manager.**TalonLRUComponentManager**(**args: Any*, ***kwargs: Any*)

Bases: *CbfComponentManager*

A component manager for the TalonLRU device.

start_communicating() → None

Establish communication with the component, then start monitoring.

stop_communicating() → None

Stop communication with the component.

get_device_proxy(*fqdn: str*) → CbfDeviceProxy | None

Attempt to get a device proxy of the specified device.

Parameters

fqdn – FQDN of the device to connect to

Returns

CbfDeviceProxy to the device or None if no connection was made

check_power_mode(*state: tango.DevState*) → None

Get the power mode of both PDUs and check that it is consistent with the current device state.

Parameters
state – device operational state

on(*simulation_mode: ska_tango_base.control_model.SimulationMode*) → Tuple[ska_tango_base.commands.ResultCode, str]

Turn on the TalonLRU and its subordinate devices

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

Return type
(ResultCode, str)

off() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn off the TalonLRU and its subordinate devices

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

Return type
(ResultCode, str)

standby() → Tuple[ska_tango_base.commands.ResultCode, str]

Turn the TalonLRU into low power standby mode

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

Return type
(ResultCode, str)

6.8 TalonDxLogConsumer

6.8.1 TalonDxLogConsumer Class

class ska_mid_cbf_mcs.talondx_log_consumer.talondx_log_consumer_device.**TalonDxLogConsumer**(**args: Any*, ***kwargs: Any*)

Bases: SKABaseDevice

TANGO device class for consuming logs from the Tango devices run on the Talon boards, converting them to the SKA format, and outputting them via the logging framework.

create_component_manager()

Create the component manager LogComponentManager

Returns
Instance of LogComponentManager

Return type
LogComponentManager

Log(*log_message: List[str]*)

Write the log to stdout as received from TLS

Sample log: ['1650964795495', 'ERROR', 'ska001/elt/master', 'TangoUtils::DeviceAttributeToCorbaAny() - A Message', '', '@7f48dcc80700 [7]']

Details of the list items here: https://tango-controls.readthedocs.io/projects/rfc/en/latest/14/Logging.html#log-consumer

Parameters
log_message (*List[str]*) – Parts of the TLS log message

SetTalonDxLogConsumerTarget(*device_name: str*) → None

Add TalonDxLogConsumer as a logging target destination on device

RemoveTalonDxLogConsumerTarget(*device_name: str*) → None

Remove TalonDxLogConsumer as a logging target destination on device

write_loggingLevel(*value: ska_tango_base.control_model.LoggingLevel*)

Sets logging level for the device. Both the Python logger and the Tango logger are updated. Overrides the base class attribute to accept all log levels coming from HPS devices, but still limit the logging level of TalonDxLogConsumer logs.

Parameters
value – Logging level for logger

Raises
LoggingLevelError – for invalid value

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

G

I

J

L

M

N

O

P

Q

R

S

W

Z