LOW CSP.LMC Software Documentation

Release 0.9.0

SKA Organization

Sep 20, 2023

LOW CSP.LMC DESCRIPTION

CHAPTER

ONE

LOW.CSP LMC DOCUMENTATION

The top-level software components provided by Mid.CSP LMC API are:

- *Low.CSP LMC Controller*
- *Low.CSP LMC Subarray*

Components listed above are implemented as TANGO devices, i.e. classes that implement standard TANGO API. The CSP.LMC TANGO devices are based on the standard SKA1 TANGO Element Devices provided via the *SKA Base Classes*.

1.1 Low.CSP LMC Controller

Low.CSP Controller is the top-level TANGO Device and the primary point of contact for monitor and control of the Low.CSP Sub-syste.

The Low.CSP Controller represents Low.CSP Sub-system as a unit for control and monitoring for general operations.

Low.CSP Controller main roles are:

- To be the central control node for Low.CSP. The Controller provides a single point of access for control of the Low.CSP as a whole, this includes provision for housekeeping and supervisory commands including: power-up, power-down, power management, restart (re-initialize), support for firmware and software upgrades, etc.
- To provide rolled-up reporting for the overall Low.CSP status. Low.CSP Controller monitors and intelligently rolls-up status reported by Low.CSP equipment, sub-arrays and capabilities and maintains a standard set of states and modes, as defined in the document "SKA1 Control System Guidelines". State transitions are reported using standard TANGO mechanism.
- To implement a set of attributes that represent the status and configuration of the Low.CSP as a whole and, where required, report availability, status and configuration of the Low.CSP equipment, components and capabilities (in the form of lists or tables or JSON string).
- To maintains the pool of resources (VCCs, FSPs, Search/Timing/VLBI beams), keep track of allocation to sub-arrays and provide reports on resource availability, allocation, and more.

Low.CSP Controller implementation is based on (derived from) the standard SKA1 TANGO Device Controller; Low.CSP Controller implements the standard TANGO API, aligned with the document "SKA1 Control System Guidelines".

The interface is between a TANGO client and a TANGO Device. The TANGO Device exposes attributes and commands to clients.

The roles of the interfacing systems are:

- TANGO Clients: Low.TMC sub-system TANGO client(s).

- TANGO Device: Low.CSP sub-system implements Low.CSP Controller.

The clients use requests to obtain read and/or write access to TANGO device attributes, and to invoke TANGO device commands.

1.1.1 Low.CSP Controller TANGO Device name

The Low.CSP Controller TANGO Device name is defined in the document "SKA1 TANGO Naming Conventions":

```
low-csp/control/0
```

1.1.2 Low.CSP Controller TANGO Device Properties

The Low.CSP Controller device has a standard set of properties inherited from the SKA Controller TANGO Device and a number of specific properties documented in the Controller API section.

1.1.3 Low.CSP Controller TANGO Device States and Modes

Low.CSP Controller implements the standard set of state and mode attributes defined by the SKA Control Model.

Low.CSP Controller reports on behalf of the Low.CSP Sub-system – unless explicitly stated otherwise, the state and mode attributes implemented by the Low.CSP Controller represent the status of the Low.CSP as a whole, not the status of the Low.CSP Controller itself.

1.1.4 Low.CSP Controller operational state

The Low.CSP Controller supports the following sub-set of the TANGO Device states:

- UNKNOWN: Low.CSP is unresponsive, e.g. due to communication loss. This state cannot be reported by CSP itself.
- OFF: power is disconnected. This state cannot be reported by CSP itself
- INIT: Initialization of the monitor and control network,equipment and functionality is being performed. During initialization commands that request state transition to OFF (power-down) or re-start initialization are accepted.
- DISABLE: Low.CSP is administratively disabled, either by setting adminMode=OFFLINE or NOT-FITTED. Basic monitor and control functionality is available but signal processing functionality and related commands are not available. All sub-arrays are empty (OFF) and IDLE; all resources (receptors, tide-array beams) are placed in the pool of unused resources.)
- STANDBY: Low-power state, Low.CSP uses < 5% of nominal power. Basic monitor and control functionality is available, including the commands to request state transition to ON, OFF, DISABLE, or INIT. Signal processing functionality and related commands are not available. All sub-arrays are empty (OFF) and IDLE; all resources (receptors, tide-array beams) are placed in the pool of unused resources.).
- ON: At least a minimum of CSP signal processing capability is available; at least one receptor and one sub-array can be used for observing (either for scientific observations or for testing andmaintenance). Low.CSP is in normal operational state, all commands, including commands to increase/decrease functional availability and power consumption are available.
- ALARM: Quality Factor for at least one attribute crossed the ALARM threshold. Part of Low.CSP functionality may be unavailable.

- FAULT: Unrecoverable fault has been detected, Low.CSP is not available for use at all, maintainer/operator intervention is required in order to return to ON, STANDBY, or DISABLE. Depending on the extent of failure commands restart and init, as well as status reporting may be available.

1.1.5 Low.CSP Controller TANGO Device Commands

Low.CSP Controller implements the standard set of commands as specified in:

- Standard set of TANGO Device commands as defined in TANGO User Manual
- Standard set of SKA TANGO Device commands
- Command specific to Low.CSP Controller as described in API section.

Low.CSP makes provision for TM to request state transitions for individual sub-systems and/or Capabilities.

1.1.6 Low.CSP Controller TANGO Device Attributes

Low.CSP Controller implements the standard set of attributes as specified in:

- Standard set of TANGO Device attributes as defined in TANGO User Manual
- The standard set of SKA TANGO Device attributes as defined for the SKA Controller TANGO Device.
- Attributes to Low.CSP Controller as described in API section.

The Low.CSP Controller maintains the 'pool of resources' and is able to provide information regarding sub-array membership, status and usage.

1.2 Low.CSP LMC Subarray

The core CSP functionality, configuration and execution of signal processing, is configured, controlled and monitored via subarrays.

CSP Subarray makes provision to TM to configure a subarray, select Processing Mode and related parameters, specify when to start/stop signal processing and/or generation of output products. TM accesses directly a CSP Subarray to:

- Assign resources
- Configure a scan
- Control and monitor states/operations

The assignment of Capabilities to a subarray (*subarray composition*) is performed in advance of a scan configuration. Assignable Capabilities for LOW CSP.LMC Subarrays are:

- stations
- station beams
- tied-array beams: Search Beams, Timing Beams and Vlbi Beams.

In general resource assignment to a subarray is exclusive, but in some cases the same Capability instance may be used in shared manner by more then one subarray.

1.2.1 Inherent Capabilities

Each CSP subarray has also five permanently assigned *inherent Capabilities*:

- Station Beam
- Correlation
- PSS
- PST
- VLBI

An inherent Capability can be enabled or disabled, but cannot assigned or removed to/from a subarray. They correspond to the CSP Low Processing Modes and are configured via a scan configuration.

Scan configuration

TM provides a complete scan configuration to a subarray via an ASCII JSON encoded string. Parameters specified via a JSON string are implemented as TANGO Device attributes and can be accessed and modified directly using the buil-in TANGO method *write_attribute*. When a complete and coherent scan configuration is received and the subarray configuration (or re-configuration) completed, the subarray it's ready to observe.

Control and Monitoring

Each Low CSP.LMC Subarray maintains and reports the status and state transitions for the Low CSP subarray as a whole and for the individual assigned resources.

In addition to pre-configured status reporting, a Low CSP Subarray makes provision for the TM and any authorized client, to obtain the value of any subarray attribute.

1.2.2 Low.CSP Subarray TANGO Device name

The Low.CSP Subarray TANGO Device name is defined in the document "SKA1 TANGO Naming Conventions":

```
low-csp/subarray/XY
```

where XY is a two digit number in range [01,..,16].

1.2.3 Low.CSP Subarray operational state

Low.CSP Subarray intelligently rolls-up the operational state of all components used by the sub-array and reports the overall operational state for the sub-array.

The Low.CSP Subarray supports the following sub-set of the TANGO Device states:

- UNKNOWN: Low.CSP sub-array is unresponsive, e.g. due to communication loss.
- OFF: The sub-array is not enabled to perform signal processing. The sub-array is 'empty'.
- INIT: Initialization of the monitor and control network,equipment and functionality is being performed.
- DISABLE: Low.CSP sub-array is administratively disabled, basic monitor and control functionality is available but signal processing functionality is not available.
- ON: The sub-array is enabled to perform signal processing. The sub-array observing state is EMPTY if receptors have not been assigned to the sub-array, yet.

- ALARM: Quality Factor for at least one attribute crossed the ALARM threshold. Part of functionality may be unavailable.
- FAULT: Unrecoverable fault has been detected. The sub-array is not available for use and maintainer/operator intervention might be required.

1.2.4 Low.CSP Subarray observing state

The sub-array Observing State indicates status related to scan configuration and execution.

The Low.CSP Subarray observing state adheres to the State Machine defined by ADR-8.

1.2.5 Low.CSP Subarray TANGO Device Commands

Low.CSP Subarray implements the standard set of commands as specified in:

- Standard set of TANGO Device commands as defined in TANGO User Manual
- Standard set of SKA TANGO Device commands
- Command specific to Low.CSP Subarray as described in API section.

Low.CSP makes provision for TM to request state transitions for individual sub-systems and/or Capabilities.

1.2.6 Low.CSP Subarray TANGO Device Attributes

Low.CSP Subarray implements the standard set of attributes as specified in:

- Standard set of TANGO Device attributes as defined in TANGO User Manual
- The standard set of SKA TANGO Device attributes as defined for the SKA Subarray TANGO Device.
- Attributes to Low.CSP Subarray as described in API section.

Virtually all parameters provided in scan configuration are exposed as attributes either by Low.CSP Subarray or by individual Capabilities.

CHAPTER

TWO

LOW CSP.LMC ARCHITECTURE

The architecture of the CSP.LMC software is the same for Low and Mid Telescope. Please refer to common documentation

CHAPTER

THREE

PROJECT'S API

Below are presented the API for Classes specialized for Low Telescope.

All the functionalities common to Mid and Low Telescope are developed in the project ska-csp-lmc-common

3.1 Low.CSP LMC Devices API

3.1.1 Low CSP Controller API

3.1.2 Low CSP Subarray API

3.2 Low.CSP LMC modules API

3.2.1 Low.CSP Sub-system Components

Components class work as adapters and caches towards the Low.CSP sub-systems TANGO devices.

Low.CBF Controller Component

class ska_csp_lmc_low.controller.low_ctrl_component.**LowCbfControllerComponent**(**args: Any*, ***kwargs: Any*)

Bases: `CbfControllerComponent`

Specialize the CBF Controller Component class for the Low.CSP LMC.

This class works as a cache and adaptor towards the real Low device.

__init__(*fqdn: str*, *logger: Optional[Logger] = None*)

property list_of_stations

Return the list of receptors IDs.

Low.PSS Controller Component

class ska_csp_lmc_low.controller.low_ctrl_component.**LowPssControllerComponent**(**args: Any*, ***kwargs: Any*)

Bases: PssControllerComponent

Specialization of the PssController component class for the Low.CSP.

__init__(*fqdn: str*, *logger: Optional[Logger] = None*)

property list_of_beams

Return the list of PSS beams IDs.

Low.CBF Subarray Component

class ska_csp_lmc_low.subarray.low_subarray_component.**LowCbfSubarrayComponent**(**args: Any*, ***kwargs: Any*)

Bases: CbfSubarrayComponent

Specialization of the Cbf Subarray component class for the LOW Telescope.

property assigned_stations: List[int]

stations assigned to Cbf subarray on success, otherwise an empty list

Type
return

property assigned_station_beams: List[int]

stations beams currently assigned to Cbf subarray on success, otherwise an empty list

Type
return

property assigned_pss_beams: List[int]

pss Beams assigned to Cbf subarray on success, otherwise an empty list

Type
return

property assigned_pst_beams: List[int]

pst Beams currently assigned to Cbf subarray on success, otherwise an empty list

Type
return

releaseresources(*resources_list*, *callback: Callable*) → None

Invoke the release resources command on the Low CBF subarray.

Parameters

- **resources_list** – The release resource dictionary with configuration
- **callback** – Method invoked when the commands end on the target device

Raise
ValueError exception if the list of receptors specified into the configuration dictionary is not valid

assignresources(*resources_list*, *callback*)

Invoke the assign resources command on the Low CBF subarray.

Parameters

- **argument** – The assign resource dictionary with configuration
- **callback** – Method invoked when the commands end on the target device

Raise

ValueError exception if the list of receptors specified into the configuration dictionary is not valid

validated_resources(*resources: dict*, *action: str*)

pylint: disable-next=fixme TODO: available resources of CBF are required to implement this method.

releaseallresources(*callback: Optional[Callable] = None*)

Invoke the releaseall resources command on the Low CBF subarray.

Parameters

callback – Method invoked when the commands end on the target device

Raise

ValueError exception if the list of receptors specified into the configuration dictionary is not valid

configure(*resources: dict*, *callback: Optional[Callable] = None*)

Invoke the releaseall resources command on the Low CBF subarray.

Parameters

callback – Method invoked when the commands end on the target device

Raise

ValueError exception if the list of receptors specified into the configuration dictionary is not valid

gotoidle(*callback: Optional[Callable] = None*)

Invoke the releaseall resources command on the Low CBF subarray.

Parameters

callback – Method invoked when the commands end on the target device

Raise

ValueError exception if the list of receptors specified into the configuration dictionary is not valid

Low.PSS Subarray Component

class ska_csp_lmc_low.subarray.low_subarray_component.**LowPssSubarrayComponent**(**args: Any*, ***kwargs: Any*)

Bases: PssSubarrayComponent

Specialization of the PssSubarray component class for the Low Telescope.

validated_resources(*resources: dict*, *action: str*) → List[int]

Validate the input configuration for the resources to assign.

Parameters

- **resources** – the PSS resources to assign/remove to/from the Low Subarray

- **action** – the action to perform: assign or release

assignresources(*resources_list: Dict*, *callback: Optional[Callable] = None*) → None

Invoke the assign resources command on the Low PSS subarray.

Parameters

- **resources_list** – The assign resource dictionary with configuration
- **callback** – Method invoked when the commands end on the target device

Returns

None

Raise

ValueError exception if the list of receptors specified into the configuration dictionary is not valid

releaseallresources(*callback: Optional[Callable] = None*)

Invoke the releaseall resources command on the Low PSS subarray.

Parameters

callback – Method invoked when the commands end on the target device

Returns

None

Raise

ValueError exception if the list of receptors specified into the configuration dictionary is not valid

CHAPTER

FOUR

TANGO CLIENTS EXAMPLES

Basic assumptions for each example are:

1. the system has been fresh initialized
2. the only CSP sub-system deployed is the CBF
3. All TANGO operations (read/write/command_inout) are always successfully. No check on the results is done in the following examples.

To get instruction on how to deploy a working system, please refer to the project's README.

Be sure that both the charts for `ska-low-cbf` and `ska-low-cbf-proc`, as stated in the 'low-umbrella-chart' are deployed.

To control CSP with itango, a proxy to CSP Controller and Subarrays has to be created:

```
csp_ctrl = tango.DeviceProxy('low-csp/control/0')
csp_sub1 = tango.DeviceProxy('low-csp/subarray/01')
csp_sub2 = tango.DeviceProxy('low-csp/subarray/02')
csp_sub3 = tango.DeviceProxy('low-csp/subarray/03')
```

It is possible also to create proxies to CBF controller and subarrays, in order to check their states and mode after the command is issued:

```
cbf_ctrl = tango.DeviceProxy('low-cbf/control/0')
cbf_sub1 = tango.DeviceProxy('low-cbf/subarray/01')
cbf_sub2 = tango.DeviceProxy('low-cbf/subarray/02')
cbf_sub3 = tango.DeviceProxy('low-cbf/subarray/03')
```

Please note that the Low CBF is currently deploying 4 subarrays. If a Low CBF subarray is not deployed, the corresponding Low CSP.LMC subarray state is FAULT and can't be used for operations.

Also note that all commands on subarray follow the ObsState state model defined in ADR-8. CSP/LMC doesn't allow commands from obstate other than those specified in this model.

From now on, all the examples refers to these proxy objects. For Subarray commands, the proxy to subarray 1 will be used.

4.1 Low CSP.LMC state after deployed

In the current deployment (see *low-csp-umbrella* to get the charts versions) Low CBF deploys only one subarray and the state and modes of the Low.CSP Controller and Subarrays after the deployment are:

```
csp_ctrl.state() = DISABLE
csp_sub1.state() = DISABLE
csp_sub2.state() = DISABLE
csp_sub3.state() = DISABLE
csp_ctrl.healthstate = UNKNOWN
csp_sub1.healthstate = UNKNOWN
csp_sub2.healthstate = UNKNOWN
csp_sub3.healthstate = UNKNOWN
csp_ctrl.adminMode = OFFLINE
csp_sub1.adminMode = OFFLINE
csp_sub2.adminMode = OFFLINE
csp_sub3.adminMode = OFFLINE
csp_sub1.obsstate = EMPTY
csp_sub2.obsstate = EMPTY
csp_sub3.obsstate = EMPTY
```

Low CSP.LMC Controller and Subarrays adminMode have to be set to MAINTENANCE or ONLINE to start the connection with the subordinate Low CBF TANGO Devices.

To start the communication use the following command:

```
csp_ctrl.adminMode = 2  #set to MAINTENANCE
```

or

```
csp_ctrl.adminMode = 0  #set to ONLINE
```

Low CSP.LMC Controller forwards the *adminMode* value to its Subarrays and subordinated systems devices. The new states are the following for all the devices:

```
csp_ctrl.state() = ON
csp_ctrl.healthstate = UNKNOWN (It will be fixed to OK in one of the next releases)
csp_sub1.state() = ON
csp_sub1.healthstate = UNKNOWN (It will be fixed to OK in one of the next releases)
csp_sub2.state() = DISABLE (It will be fixed to FAULT in one of the next releases)
csp_sub2.healthstate = UNKNOWN (It will be fixed to FAILED in one of the next releases)
csp_sub3.state() = DISABLE (It will be fixed to FAULT in one of the next releases)
csp_sub3.healthstate = UNKNOWN (It will be fixed to FAILED in one of the next releases)
cbf_ctrl.state() = ON
cbf_sub1.state() = ON
```

Low CSP.LMC Subarray 2 and 3 are in DISABLE because the corresponding Low CBF subarrays are not deployed.

4.2 Power-on (off/standby) the Low.CSP

Note: CBF Devices start in ON state and so CSP. Anyway the command ON can be still meaningful for turning on other subsystems. If only cbf is present, this section can be dropped

To power-on all the device, send the ON command towards the Low.CSP Controller. The sub-systems already in ON state will be skipped by the command.

```
csp_ctrl.On([]) # empty list: power on all the available sub-systems (CBF, PSS, PST)
```

or

```
csp_ctrl.On(['low-pst/beam/01', ]) # power-on only the specified sub-systems
```

The command returns immediately the following list:

```
[array([2], dtype=int32), ['1679401117.9451234_224758016395799_On']]
```

Where:

- 2 is the command status (2 means QUEUED)
- '1679401117.9451234_224758016395799_On' is a unique id assigned to the command. It can be used to track the execution status of the command

It is possible to read the command result state using:

```
cmd_result = csp_ctrl.commandResult
```

cmd_result is a Tuple of two strings:

- the first one is the name of last executed CSP task
- the second one is the result code: allowed values for the result code are defined in SKA Base Classes module ska_tango_base.commands

Possible results for the current example are:

- ('on', '0') # *On* task completed successfully
- ('on', '1') # *On* task started
- ('on', '3') # *On* task completed with failure

Some of the long running command attributes can also be accessed. Long running command result can be read using:

```
long_running_command_result = csp_ctrl.longRunningCommandResult
```

long_running_command_result is a Tuple of a string and a list:

- the first element is a command id assigned when command is invoked
- the second element is a list of result code (matching the value in command result attribute described above) and result message

Possible results:

- ('1684312814.139426_265125881596693_On', '[0, "on completed 1/1"]') # *On* task completed successfully on one devices

- ('1684312814.139426_265125881596693_Configure', '[0, "configure completed on components 2/2"]') # *Configure* task completed successfully on two devices
- ('1684312814.139426_265125881596693_Off', '[3, "off completed 1/1"]') # *Off* task completed with failure on one device

It is also possible to access long running command status at the various stages of command execution. This attribute can be read using:

```
long_running_command_status = csp_ctrl.longRunningCommandStatus
```

long_running_command_status is a Tuple of pairs of strings. In each pair the elements represent the following:

- the first element is a command id assigned when command is invoked
- the second element is the task status: allowed values for the task status are defined in SKA Base Classes module ska_tango_base.executor

Possible results:

- ('1684312814.139426_265125881596693_On', 'QUEUED')
- ('1684312814.139426_265125881596693_On', 'IN_PROGRESS')
- ('1684312814.139426_265125881596693_On', 'COMPLETED')
- ('1684312814.139426_265125881596693_On', 'ABORTED')
- ('1684312814.139426_265125881596693_On', 'FAILED')
- ('1684312814.139426_265125881596693_On', 'REJECTED')

Note that there can be more than one pair of command id and task status in the attribute, if there are more commands invoked.

The command *On* invoked on the Low.CSP Controller is forwarded to the CBF sub-system Controller and to all the Low.CSP Subarrays. This can be checked by the state of all controllers and subarrays:

```
csp_ctrl.state() -> ON
csp_sub1.state() -> ON
csp_sub1.state() -> ON
csp_sub1.state() -> ON

cbf_ctrl.state() -> ON
cbf_sub1.state() -> ON
cbf_sub1.state() -> ON
cbf_sub1.state() -> ON
```

The same logic and syntax apply also for Off and Standby command

4.3 Assign resources to the Low.CSP Subarray

To assign resources on a subarray both the subarray and controller are needed to be on. To do this please follow the previous example.

A `json_string` variable needs to be create containing the proper json structure for assignment of resources. A working template can be found *here*

Invoke the AssignResources command on Low.CSP Subarray 1:

```
csp_sub1.assignresources(json_string)
```

If command is successful, the command result will report:

```
csp_sub1.commandResult -> ('assignresources', '0')

csp_sub1.commandResultName -> 'assignresources'
csp_sub1.commandResultCode -> '0'
```

After the assignment of resources the CSP.LMC and CBF Subarray osbstate move from EMPTY to IDLE. It can be checked by:

```
csp_sub1.obsstate -> IDLE
cbf_sub1.obsstate -> IDLE
```

4.4 Configure, issue and end a Scan

After the Subarray has resources assigned, it is possible to configure and run a scan on the Subarray. When CBF is in simulation mode (i.e. all the deployment except th PSI-LOW) some preliminar operation are needed before configuring the subarray. In particular, a serial number has to be assigned to the low-cbf processor device and this one subscribed by the low-cbf allocator. To do this:

```
processor1 = tango.DeviceProxy("low-cbf/processor/0.0.0")
processor1.serialnumber = "XFL14SLO1LIF"
processor1.subscribetoallocator("low-cbf/allocator/0")
processor1.register()
```

If this is not performed, the configure command will fail.

The `json_string` to be used for configure and scan can be found *here*. They have to be assigned to a variable and send as a command input as showed above for assignresources.

First of all, Configure command has to be issued:

```
csp_sub1.configure(json_string_configure)
```

The obsstate will be in CONFIGURING during the execution. After that, if the command is successful:

```
csp_sub1.commandResult -> ('configure', '0')
csp_sub1.obsstate -> READY
cbf_sub1.obsstate -> READY
```

A new configuration can be sent also in READY state, overwriting the old one.

After the subarray is READY, a scan can be issued:

```
csp_sub1.scan(json_string_scan)
```

if the command is successful:

```
csp_sub1.commandResult -> ('scan', '0')
csp_sub1.obsstate -> SCANNING
cbf_sub1.obsstate -> SCANNING
```

It as been decided that the command 'scan' has to be considered as a normal command that return the status 0 when all the subsystems are moved to SCANNING status. The scan behavior is documented in Scan handling . According to ADR-8 a Scan can be interrupted by the EndScan Command or the Abort Command. The Abort Command has to be intended as an emergency call that interrupts abruptly the Scan process. On the other side, the EndScan first ensure that all the processes are correctly managed.

To end a scan, just issue:

```
csp_sub1.endscan()
```

After End Scan is successful, the ObState of subarray is READY, and another Scan can be issued with the same configuration.

On the other side, if the Scan is aborted, the obsstate will go (after a short time in ABORTING) to be ABORTED. To perform a new scanning, the subarray observation should be restarted (via the ObsReset command) and a new configuration need to be sent (ADR-8)

The sequence of operation is:

```
csp_sub1.abort()
csp_sub1.commandResult -> ('abort', '1')
csp_sub1.obsstate -> ABORTING
csp_sub1.commandResult -> ('abort', '0')
csp_sub1.obsstate -> ABORTED
csp_sub1.obsreset()
csp_sub1.commandResult -> ('obsreset', '1')
csp_sub1.obsstate -> RESETTING
csp_sub1.commandResult -> ('obsreset', '0')
csp_sub1.obsstate -> IDLE
```

4.5 Go To Idle and Release Resources

The release of the resources of a subarray can be done only in IDLE obsstate. For this reason, if the subarray is in READY firstly must be sent to IDLE with the command:

```
csp_sub1.gotoidle()
```

if the command is successful, the obsstate will be IDLE. After that, the resources can be partially or totally removed. To partially remove some resources, a json string, like the one used for assign resources (see above) should be sent. In this string, please specify the resources to be removed.

csp_sub1.releaseresources(json_string)

On command success:

```
csp_sub1.commandResult -> ('releaseresources', '0')
csp_sub1.obsstate -> IDLE
```

Otherwise, if all resources are meant to be removed, this can be done with the ReleaseAllResouces command:

```
csp_sub1.releaseallresources()
```

On command success, the subarray will be EMPTY again, and some resouces need to be added to perform a new operation:

```
csp_sub1.commandResult -> ('releaseallresources', '0')
csp_sub1.obsstate -> EMPTY

csp_sub1.assignedReceptors -> []
```

4.6 Recover from FAULT ObsState: Restart

If something goes wrong in the observation, the ObsState of CSP.LMC could go in FAULT. Please note that this not refers to the case of the Tango Device has an internal error and the *DevState* goed in FAULT (see next for this case). To recover from this situation, the restart command is issued. This command will release also all the resources, taking the subarray in an EMPTY obstate.

The sequence of operations and responses is:

```
csp_sub1.obsstate -> FAULT

csp_sub1.restart()

csp_sub1.commandResult -> ('restart', '1')
csp_sub1.obsstate -> RESTARTING
csp_sub1.commandResult -> ('restart', '0')
csp_sub1.obsstate -> EMPTY
```

4.7 Recover from FAULT DevState: Reset

Both Subarray and Controller, if experience an error internal to TANGO Device, will go in FAULT DevState. To recover from it, the Reset command needs to be issued. This will bring the device in its initial state, i.e. OFF/EMPTY for Subarray and STANDBY for the Controller

4.8 Turning OFF the subarray

The *Off* command disables any signal processing capability of a subarray and all its allocated resources are also released. As for the ADR-8, this command can be issued fron *any* observing state.

Depending on the current observing state of the Mid CSP.LMC Subarray, the *Off* command can be replaced by a sequence of commands that properly bring the device in the desired final state. An approach that works for nearly all the observing states is the following one, where the *Off* command is replaced by the following commands, executed one after the other:

- *Abort*: transition the subarray from the current observing state to ABORTED. This command can be issued from all the observing states except: EMPTY

and FAULT. In these cases, this step is skipped and the first command invoked must be *Restart*. * Restart: transition the subarray from ABORTED to EMPTY/ON * Off: transition the subarray from ON to OFF.

CHAPTER

FIVE

JSON COMMAND INPUT TEMPLATES

The following templates can be used as input for the specific command. They are those used in Low CSP.LMC integration tests.

5.1 Assign Resources

```
{
    "interface": "https://schema.skao.int/ska-low-csp-assignresources/2.0",
    "common": {
        "subarray_id": 1
    },
    "lowcbf":{},
    "lowpss": {
        "beams_id":[1 ,2 ,3]
    }
}
```

5.2 Configure

```
{
```

“interface”: “https://schema.skao.int/ska-low-csp-configurescan/0.0”,

“subarray”: {
“subarray_name”: “science period 23”

}, “common”: {

“config_id”: “sbi-mvp01-20200325-00001-science_A”, “subarray_id”: 1, “frequency_band”: “low”

}, “lowcbf”: {

“stations”: {
“stns”: [[1, 1], [2, 1], [3, 1], [4, 1], [5, 1], [6, 1]], “stn_beams”: [

{
“beam_id”: 1, “freq_ids”: [400], “delay_poly”: “tango://delays.skao.int/low/stn-beam/1”

}

```
            ]
        }, "vis": {
            "fsp": {"firmware": "vis", "fsp_ids": [1]}, "stn_beams": [
              {
                "stn_beam_id": 1, "host": [[0, "192.168.0.1"]], "port": [[0, 9000, 1]],
                "mac": [[0, "02-03-04-0a-0b-0c"]], "integration_ms": 849
              }
            ]
        }
    }
}
```


5.3 Scan


```
{
    "interface": "https://schema.skao.int/ska-low-csp-scan/0.0", "common": {
        "subarray_id": 1
    }, "lowcbf": {
        "scan_id":987654321
    }
}
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols