SKA Tango Base Documentation

Release 0.9.1

NCRA India

Mar 02, 2021

STANDARD DEVICES

CHAPTER

ONE

BASE DEVICE

This module implements a generic base model and device for SKA. It exposes the generic attributes, properties and commands of an SKA device.

1.1 Tango Device Class

class ska_tango_base.**SKABaseDevice**(**args: Any*, ***kwargs: Any*)

A generic base device for SKA.

class InitCommand(*target*, *state_model*, *logger=None*)

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

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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)

succeeded()

Callback for the successful completion of the command.

SkaLevel = tango.server.device_property(dtype=int16, default_value=4)

Device property.

Indication of importance of the device in the SKA hierarchy to support drill-down navigation: 1..6, with 1 highest.

GroupDefinitions = tango.server.device_property(dtype=('str',))

Device property.

Each string in the list is a JSON serialised dict defining the `group_name`, `devices` and `subgroups` in the group. A TANGO Group object is created for each item in the list, according to the hierarchy defined. This provides easy access to the managed devices in bulk, or individually.

The general format of the list is as follows, with optional `devices` and `subgroups` keys:

```
[ {"group_name": "<name>",
   "devices": ["<dev name>", ...]},
  {"group_name": "<name>",
   "devices": ["<dev name>", "<dev name>", ...],
   "subgroups" : [{<nested group>},
                  {<nested group>}, ...]},
  ...
  ]
```

For example, a hierarchy of racks, servers and switches:

```
[ {"group_name": "servers",
   "devices": ["elt/server/1", "elt/server/2",
               "elt/server/3", "elt/server/4"]},
  {"group_name": "switches",
   "devices": ["elt/switch/A", "elt/switch/B"]},
  {"group_name": "pdus",
   "devices": ["elt/pdu/rackA", "elt/pdu/rackB"]},
  {"group_name": "racks",
   "subgroups": [
        {"group_name": "rackA",
         "devices": ["elt/server/1", "elt/server/2",
                     "elt/switch/A", "elt/pdu/rackA"]},
        {"group_name": "rackB",
         "devices": ["elt/server/3", "elt/server/4",
                     "elt/switch/B", "elt/pdu/rackB"],
         "subgroups": []}
   ]} ]
```

LoggingLevelDefault = tango.server.device_property(dtype=uint16, default_value=4)
Device property.

Default logging level at device startup. See *LoggingLevel*

LoggingTargetsDefault = tango.server.device_property(dtype=DevVarStringArray, default_
Device property.

Default logging targets at device startup. See the project readme for details.

buildState = tango.server.attribute(dtype=str, doc=Build state of this device)
Device attribute.

versionId = tango.server.attribute(dtype=str, doc=Version Id of this device)
Device attribute.

loggingLevel = tango.server.attribute(dtype=<enum 'LoggingLevel'>, access=tango.AttrWr
Device attribute.

See *LoggingLevel*

loggingTargets = tango.server.attribute(dtype=('str',), access=tango.AttrWriteType.REA
Device attribute.

healthState = tango.server.attribute(dtype=<enum 'HealthState'>, doc=The health state
Device attribute.

adminMode = tango.server.attribute(dtype=<enum 'AdminMode'>, access=tango.AttrWriteTyp
Device attribute.

controlMode = tango.server.attribute(dtype=<enum 'ControlMode'>, access=tango.AttrWrit
Device attribute.

simulationMode = tango.server.attribute(dtype=<enum 'SimulationMode'>, access=tango.At
Device attribute.

testMode = tango.server.attribute(dtype=<enum 'TestMode'>, access=tango.AttrWriteType.
Device attribute.

set_state(*state*)
Helper method for setting device state, ensuring that change events are pushed.

Parameters **state** (tango.DevState) – the new state

set_status(*status*)
Helper method for setting device status, ensuring that change events are pushed.

Parameters **status** (*str*) – the new status

init_device()
Initializes the tango device after startup.

Subclasses that have no need to override the default default implementation of state management may leave init_device() alone. Override the do() method on the nested class InitCommand instead.

register_command_object(*command_name*, *command_object*)
Registers a command object as the object to handle invocations of a given command

Parameters

- **command_name** (*str*) – name of the command for which the object is being registered
- **command_object** (*Command instance*) – the object that will handle invocations of the given command

get_command_object(*command_name*)
Returns the command object (handler) for a given command.

Parameters **command_name** (*str*) – name of the command for which a command object (handler) is sought

Returns the registered command object (handler) for the command

Return type Command instance

init_command_objects()
Creates and registers command objects (handlers) for the commands supported by this device.

always_executed_hook()
Method that is always executed before any device command gets executed.

delete_device()
Method to cleanup when device is stopped.

read_buildState()
Reads the Build State of the device.

Returns the build state of the device

read_versionId()
Reads the Version Id of the device.

Returns the version id of the device

read_loggingLevel()
Reads logging level of the device.

Returns Logging level of the device.

write_loggingLevel(*value*)
Sets logging level for the device. Both the Python logger and the Tango logger are updated.

Parameters **value** – Logging level for logger

Raises **LoggingLevelError** – for invalid value

read_loggingTargets()
Reads the additional logging targets of the device.

Note that this excludes the handlers provided by the ska_ser_logging library defaults.

Returns Logging level of the device.

write_loggingTargets(*value*)
Sets the additional logging targets for the device.

Note that this excludes the handlers provided by the ska_ser_logging library defaults.

Parameters **value** – Logging targets for logger

read_healthState()
Reads Health State of the device.

Returns Health State of the device

read_adminMode()
Reads Admin Mode of the device.

Returns Admin Mode of the device

Return type *AdminMode*

write_adminMode(*value*)
Sets Admin Mode of the device.

Parameters **value** (*AdminMode*) – Admin Mode of the device.

Raises **ValueError** – for unknown adminMode

read_controlMode()
Reads Control Mode of the device.

Returns Control Mode of the device

write_controlMode(*value*)
Sets Control Mode of the device.

Parameters **value** – Control mode value

read_simulationMode()
Reads Simulation Mode of the device.

Returns Simulation Mode of the device.

write_simulationMode(*value*)
Sets Simulation Mode of the device

Parameters **value** – SimulationMode

read_testMode()
Reads Test Mode of the device.

Returns Test Mode of the device

write_testMode(*value*)
Sets Test Mode of the device.

Parameters **value** – Test Mode

class GetVersionInfoCommand(*target*, *state_model*, *logger=None*)
A class for the SKABaseDevice's Reset() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

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

GetVersionInfo()
Returns the version information of the device.

To modify behaviour for this command, modify the do() method of the command class.

Returns Version details of the device.

class ResetCommand(*target*, *state_model*, *logger=None*)
A class for the SKABaseDevice's Reset() command.

Create a new ResetCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

check_allowed()
Checks whether the command is allowed to be run in the current state of the state model.
Returns True if the command is allowed to be run

is_allowed()
Whether this command is allowed to run in the current state of the state model.
Returns whether this command is allowed to run
Return type boolean

succeeded()
Action to take on successful completion of a reset

do()
Stateless hook for device reset.
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_Reset_allowed()
Whether the `Reset()` command is allowed to be run in the current state

Returns whether the `Reset()` command is allowed to be run in the current state

Return type boolean

Reset()
Reset the device from the FAULT state.

To modify behaviour for this command, modify the do() method of the command class.

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 DisableCommand(*target*, *state_model*, *logger=None*)
A class for the SKABaseDevice's Disable() command.

Constructor for DisableCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKABaseDevice for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
Stateless hook for Disable() 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_Disable_allowed()
Check if command Disable is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Disable()
Put the device into disabled mode

To modify behaviour for this command, modify the do() method of the command class.

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(*target*, *state_model*, *logger=None*)
A class for the SKABaseDevice's Standby() command.

Constructor for StandbyCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKABaseDevice for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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)

is_Standby_allowed()
Check if command Standby is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Standby()
Put the device into standby mode

To modify behaviour for this command, modify the do() method of the command class.

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(*target*, *state_model*, *logger=None*)
A class for the SKABaseDevice's Off() command.

Constructor for OffCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKABaseDevice for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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)

is_Off_allowed()
Check if command *Off* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Off()
Turn the device off

To modify behaviour for this command, modify the do() method of the command class.

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(*target*, *state_model*, *logger=None*)
A class for the SKABaseDevice's On() command.

Constructor for OnCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKABaseDevice for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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)

is_On_allowed()
Check if command *On* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

On()
Turn device on

To modify behaviour for this command, modify the do() method of the command class.

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

Return type (*ResultCode*, str)

1.2 Device State Model

class ska_tango_base.**DeviceStateModel**(*logger*, *op_state_callback=None*, *admin_mode_callback=None*)

Implements the state model for the SKABaseDevice.

This implementation contains separate state machines for adminMode and opState. Since the two are slightly but inextricably coupled, the opState machine includes "ADMIN" flavours for the "INIT", "FAULT" and "DISABLED" states, to represent states where the device has been administratively disabled via the adminModes "RESERVED", "NOT_FITTED" and "OFFLINE". This model drives the two state machines to ensure they remain coherent.

Initialises the state model.

Parameters

- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this state model.
- **op_state_callback** (*callable*) – A callback to be called when the state machine for op_state reports a change of state
- **admin_mode_callback** (*callable*) – A callback to be called when the state machine for admin_mode reports a change of state

property admin_mode

Returns the admin_mode

Returns admin_mode of this state model

Return type *AdminMode*

property op_state

Returns the op_state of this state model

Returns op_state of this state model

Return type tango.DevState

is_action_allowed(*action*)

Whether a given action is allowed in the current state.

Parameters **action** (*str*) – an action, as given in the transitions table

Raises **StateModelError** – if the action is unknown to the state machine

Returns whether the action is allowed in the current state

Return type bool

try_action(*action*)

Checks whether a given action is allowed in the current state, and raises a StateModelError if it is not.

Parameters **action** (*str*) – an action, as given in the transitions table

Raises **StateModelError** – if the action is not allowed in the current state

Returns True if the action is allowed

Return type boolean

perform_action(*action*)

Performs an action on the state model

Parameters **action** (*ANY*) – an action, as given in the transitions table

Raises `StateModelError` – if the action is not allowed in the current state

CHAPTER

TWO

ALARM HANDLER

This module implements SKAAlarmHandler, a generic base device for Alarms for SKA. It exposes SKA alarms and SKA alerts as TANGO attributes. SKA Alarms and SKA/Element Alerts are rules-based configurable conditions that can be defined over multiple attribute values and quality factors, and are separate from the "built-in" TANGO attribute alarms.

2.1 Tango Device Class

class ska_tango_base.**SKAAlarmHandler**(**args: Any*, ***kwargs: Any*)

A generic base device for Alarms for SKA.

SubAlarmHandlers = tango.server.device_property(dtype=('str',))

AlarmConfigFile = tango.server.device_property(dtype=str)

statsNrAlerts = tango.server.attribute(dtype=int, doc=Number of active Alerts)

Device attribute.

statsNrAlarms = tango.server.attribute(dtype=int, doc=Number of active Alarms)

Device attribute.

statsNrNewAlarms = tango.server.attribute(dtype=int, doc=Number of New active alarms)

Device attribute.

statsNrUnackAlarms = tango.server.attribute(dtype=double, doc=Number of unacknowledged

Device attribute.

statsNrRtnAlarms = tango.server.attribute(dtype=double, doc=Number of returned alarms)

Device attribute.

activeAlerts = tango.server.attribute(dtype=('str',), max_dim_x=10000, doc=List of act

Device attribute.

activeAlarms = tango.server.attribute(dtype=('str',), max_dim_x=10000, doc=List of act

Device attribute.

init_command_objects()

Sets up the command objects

always_executed_hook()

Method that is always executed before any device command gets executed.

delete_device()

Method to cleanup when device is stopped.

read_statsNrAlerts()

Reads number of active alerts. :return: Number of active alerts

read_statsNrAlarms()
Reads number of active alarms. :return: Number of active alarms

read_statsNrNewAlarms()
Reads number of new active alarms. :return: Number of new active alarms

read_statsNrUnackAlarms()
Reads number of unacknowledged alarms. :return: Number of unacknowledged alarms.

read_statsNrRtnAlarms()
Reads number of returned alarms. :return: Number of returned alarms

read_activeAlerts()
Reads list of active alerts. :return: List of active alerts

read_activeAlarms()
Reads list of active alarms. :return: List of active alarms

class GetAlarmRuleCommand(*target*, *state_model*, *logger=None*)
A class for the SKAAlarmHandler's GetAlarmRule() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for SKAAlarmHandler GetAlarmRule() command.
Returns Alarm configuration info: rule, actions, etc.
Return type JSON string

class GetAlarmDataCommand(*target*, *state_model*, *logger=None*)
A class for the SKAAlarmHandler's GetAlarmData() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for SKAAlarmHandler GetAlarmData() command.
Returns Alarm data
Return type JSON string

class GetAlarmAdditionalInfoCommand(*target*, *state_model*, *logger=None*)
A class for the SKAAlarmHandler's GetAlarmAdditionalInfo() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for SKAAlarmHandler GetAlarmAdditionalInfo() command.
Returns Alarm additional info
Return type JSON string

class GetAlarmStatsCommand(*target*, *state_model*, *logger=None*)
A class for the SKAAlarmHandler's GetAlarmStats() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
Stateless hook for SKAAlarmHandler GetAlarmStats() command.
Returns Alarm stats
Return type JSON string

class GetAlertStatsCommand(*target*, *state_model*, *logger=None*)
A class for the SKAAlarmHandler's GetAlertStats() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
Stateless hook for SKAAlarmHandler GetAlertStats() command.
Returns Alert stats
Return type JSON string

GetAlarmRule(*argin*)
Get all configuration info of the alarm, e.g. rule, defined action, etc.

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** – Name of the alarm

Returns JSON string containing configuration information of the alarm

GetAlarmData(*argin*)
Get list of current value, quality factor and status of all attributes participating in the alarm rule.

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** – Name of the alarm

Returns JSON string containing alarm data

GetAlarmAdditionalInfo(*argin*)
Get additional alarm information.

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** – Name of the alarm

Returns JSON string containing additional alarm information

GetAlarmStats()
Get current alarm stats.

To modify behaviour for this command, modify the do() method of the command class.

Returns JSON string containing alarm statistics

GetAlertStats()
Get current alert stats.

To modify behaviour for this command, modify the do() method of the command class.

Returns JSON string containing alert statistics

CHAPTER

THREE

LOGGER

This module implements SKALogger device, a generic base device for logging for SKA. It enables to view on-line logs through the TANGO Logging Services and to store logs using Python logging. It configures the log levels of remote logging for selected devices.

3.1 Tango Device Class

class ska_tango_base.**SKALogger**(**args: Any*, ***kwargs: Any*)

A generic base device for Logging for SKA.

init_command_objects()

Sets up the command objects

always_executed_hook()

Method that is always executed before any device command gets executed.

delete_device()

Method to cleanup when device is stopped.

class SetLoggingLevelCommand(*target*, *state_model*, *logger=None*)

A class for the SKALoggerDevice's SetLoggingLevel() command.

Constructor for SetLoggingLevelCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for SetLoggingLevel() 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)

SetLoggingLevel(*argin*)

Sets logging level of the specified devices.

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** (tango.DevVarLongStringArray) – Array consisting of

- argin[0]: list of DevLong. Desired logging level.
- argin[1]: list of DevString. Desired tango device.

Returns None.

CHAPTER

FOUR

MASTER

SKAMaster

Master device

4.1 Tango Device Class

class ska_tango_base.**SKAMaster**(**args: Any*, ***kwargs: Any*)

Master device

init_command_objects()

Sets up the command objects

class InitCommand(*target*, *state_model*, *logger=None*)

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

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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)

MaxCapabilities = tango.server.device_property(dtype=('str',))

elementLoggerAddress = tango.server.attribute(dtype=str, doc=FQDN of Element Logger)

Device attribute.

elementAlarmAddress = tango.server.attribute(dtype=str, doc=FQDN of Element Alarm Hand

Device attribute.

elementTelStateAddress = tango.server.attribute(dtype=str, doc=FQDN of Element TelStat

Device attribute.

elementDatabaseAddress = tango.server.attribute(dtype=str, doc=FQDN of Element Databas
Device attribute.

maxCapabilities = tango.server.attribute(dtype=('str',), max_dim_x=20, doc=Maximum numb
Device attribute.

availableCapabilities = tango.server.attribute(dtype=('str',), max_dim_x=20, doc=A list
Device attribute.

always_executed_hook()
Method that is always executed before any device command gets executed.

delete_device()
Method to cleanup when device is stopped.

read_elementLoggerAddress()
Reads FQDN of Element Logger device

read_elementAlarmAddress()
Reads FQDN of Element Alarm device

read_elementTelStateAddress()
Reads FQDN of Element TelState device

read_elementDatabaseAddress()
Reads FQDN of Element Database device

read_maxCapabilities()
Reads maximum number of instances of each capability type

read_availableCapabilities()
Reads list of available number of instances of each capability type

class IsCapabilityAchievableCommand(*target*, *state_model*, *logger=None*)
A class for the SKAMaster's IsCapabilityAchievable() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for device IsCapabilityAchievable() command.
Returns Whether the capability is achievable
Return type bool

isCapabilityAchievable(*argin*)
Checks of provided capabilities can be achieved by the resource(s).

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** (tango.DevVarLongStringArray.) – An array consisting pair of

- [nrInstances]: DevLong. Number of instances of the capability.

- [Capability types]: DevString. Type of capability.

Returns True if capability can be achieved, False if cannot

Return type DevBoolean

CHAPTER

FIVE

TELESCOPE STATE

SKATelState

A generic base device for Telescope State for SKA.

5.1 Tango Device Class

class ska_tango_base.**SKATelState**(**args: Any*, ***kwargs: Any*)

A generic base device for Telescope State for SKA.

TelStateConfigFile = tango.server.device_property(dtype=str)

always_executed_hook()

Method that is always executed before any device command gets executed.

delete_device()

Method to cleanup when device is stopped.

CHAPTER

SIX

OBSERVATION DEVICE

SKAObsDevice

A generic base device for Observations for SKA. It inherits SKABaseDevice class. Any device implementing an obsMode will inherit from SKAObsDevice instead of just SKABaseDevice.

6.1 Tango Device Class

class ska_tango_base.**SKAObsDevice**(**args: Any*, ***kwargs: Any*)

A generic base device for Observations for SKA.

class InitCommand(*target*, *state_model*, *logger=None*)

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

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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)

obsState = tango.server.attribute(dtype=<enum 'ObsState'>, doc=Observing State)

Device attribute.

obsMode = tango.server.attribute(dtype=<enum 'ObsMode'>, doc=Observing Mode)

Device attribute.

configurationProgress = tango.server.attribute(dtype=uint16, unit=%, max_value=100, mi

Device attribute.

configurationDelayExpected = tango.server.attribute(dtype=uint16, unit=seconds, doc=Co

Device attribute.

always_executed_hook()
Method that is always executed before any device command gets executed.

Returns None

delete_device()
Method to cleanup when device is stopped.

Returns None

read_obsState()
Reads Observation State of the device

read_obsMode()
Reads Observation Mode of the device

read_configurationProgress()
Reads percentage configuration progress of the device

read_configurationDelayExpected()
Reads expected Configuration Delay in seconds

6.2 Device State Model

class ska_tango_base.**ObsDeviceStateModel**(*action_breakdown*, *obs_machine_class*, *logger*, *op_state_callback=None*, *admin_mode_callback=None*, *obs_state_callback=None*)

Base class for ObsDevice state models

Initialises the model.

Parameters

- **action_breakdown** (*dictionary defining actions to be performed on the observation state machine and,as needed, on the device state machine.*) – the action breakdown associated with the observing state machine class
- **obs_machine_class** (transitions.Machine) – state machine for the observing state of a SKAObsDevice class device.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this state model.
- **op_state_callback** (*callable*) – A callback to be called when a transition implies a change to op state
- **admin_mode_callback** (*callable*) – A callback to be called when a transition causes a change to device admin_mode
- **obs_state_callback** (*callable*) – A callback to be called when a transition causes a change to device obs_state

property obs_state
Returns the obs_state

Returns obs_state of this state model

Return type *ObsState*

is_action_allowed(*action*)
Whether a given action is allowed in the current state.

Parameters **action** (*ANY*) – an action, as given in the transitions table

Returns where the action is allowed in the current state:

Return type bool: True if the action is allowed, False if it is not allowed

Raises **StateModelError** – for an unrecognised action

try_action(*action*)
Checks whether a given action is allowed in the current state, and raises a StateModelError if it is not.

Parameters **action** (*str*) – an action, as given in the transitions table

Raises **StateModelError** – if the action is not allowed in the current state

Returns True if the action is allowed

Return type boolean

perform_action(*action*)
Performs an action on the state model

Parameters **action** (*ANY*) – an action, as given in the transitions table

Raises **StateModelError** – if the action is not allowed in the current state

CHAPTER

SEVEN

CAPABILITY

SKACapability

Capability handling device

7.1 Tango Device Class

class ska_tango_base.**SKACapability**(**args: Any*, ***kwargs: Any*)

A Subarray handling device. It exposes the instances of configured capabilities.

init_command_objects()

Sets up the command objects

class InitCommand(*target*, *state_model*, *logger=None*)

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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)

CapType = tango.server.device_property(dtype=str)

CapID = tango.server.device_property(dtype=str)

subID = tango.server.device_property(dtype=str)

activationTime = tango.server.attribute(dtype=double, unit=s, standard_unit=s, display

Device attribute.

configuredInstances = tango.server.attribute(dtype=uint16, doc=Number of instances of

Device attribute.

usedComponents = tango.server.attribute(dtype=('str',), max_dim_x=100, doc=A list of c
Device attribute.

always_executed_hook()
Method that is always executed before any device command gets executed.

Returns None

delete_device()
Method to cleanup when device is stopped.

Returns None

read_activationTime()
Reads time of activation since Unix epoch. :return: Activation time in seconds

read_configuredInstances()
Reads the number of instances of a capability in the subarray :return: The number of configured instances of a capability in a subarray

read_usedComponents()
Reads the list of components with no. of instances in use on this Capability :return: The number of components currently in use.

class ConfigureInstancesCommand(*target*, *state_model*, *logger=None*)
A class for the SKALoggerDevice's SetLoggingLevel() command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for ConfigureInstances()) 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)

ConfigureInstances(*argin*)
This function indicates how many number of instances of the current capacity should to be configured.

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** – Number of instances to configure

Returns None.

CHAPTER

EIGHT

SUBARRAY

SKASubarray

A SubArray handling device. It allows the assigning/releasing of resources into/from Subarray, configuring capabilities, and exposes the related information like assigned resources, configured capabilities, etc.

8.1 Device Class

class ska_tango_base.**SKASubarray**(**args: Any*, ***kwargs: Any*)

Implements the SKA SubArray device

class InitCommand(*target*, *state_model*, *logger=None*)

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

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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 AssignResourcesCommand(*target*, *state_model*, *logger=None*)

A class for SKASubarray's AssignResources() command.

Constructor for AssignResourcesCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.

- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for AssignResources() command functionality.
Parameters **argin** (*list of str*) – The resources to be assigned
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 ReleaseResourcesCommand(*target*, *state_model*, *logger=None*)
A class for SKASubarray's ReleaseResources() command.

Constructor for OnCommand()

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
Stateless hook for ReleaseResources() command functionality.
Parameters **argin** (*list of str*) – The resources to be released
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 ReleaseAllResourcesCommand(*target*, *state_model*, *logger=None*)
A class for SKASubarray's ReleaseAllResources() command.

Constructor for OnCommand()

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
Stateless hook for ReleaseAllResources() 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 ConfigureCommand(*target*, *state_model*, *logger=None*)
A class for SKASubarray's Configure() command.

Constructor for ConfigureCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for Configure() command functionality.

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

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 ScanCommand(*target*, *state_model*, *logger=None*)

A class for SKASubarray's Scan() command.

Constructor for ScanCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for Scan() command functionality.

Parameters **argin** (*str*) – Scan info

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(*target*, *state_model*, *logger=None*)

A class for SKASubarray's EndScan() command.

Constructor for EndScanCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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 EndCommand(*target*, *state_model*, *logger=None*)
A class for SKASubarray's End() command.

Constructor for EndCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
Stateless hook for End() 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(*target*, *state_model*, *logger=None*)
A class for SKASubarray's Abort() command.

Constructor for AbortCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

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 ObsResetCommand(*target*, *state_model*, *logger=None*)
A class for SKASubarray's ObsReset() command.

Constructor for ObsResetCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.

- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

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 RestartCommand(*target*, *state_model*, *logger=None*)
A class for SKASubarray's Restart() command.

Constructor for RestartCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
Stateless hook for Restart() 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)

init_command_objects()
Sets up the command objects

CapabilityTypes = tango.server.device_property(dtype=('str',))

SubID = tango.server.device_property(dtype=str)

activationTime = tango.server.attribute(dtype=double, unit=s, standard_unit=s, display
Device attribute.

assignedResources = tango.server.attribute(dtype=('str',), max_dim_x=100, doc=The list
Device attribute.

configuredCapabilities = tango.server.attribute(dtype=('str',), max_dim_x=10, doc=A li
Device attribute.

always_executed_hook()
Method that is always executed before any device command gets executed.

delete_device()
Method to cleanup when device is stopped.

read_activationTime()
Reads the time since device is activated.

Returns Time of activation in seconds since Unix epoch.

read_assignedResources()
Reads the resources assigned to the device.

Returns Resources assigned to the device.

read_configuredCapabilities()
Reads capabilities configured in the Subarray.

Returns A list of capability types with no. of instances used in the Subarray

is_AssignResources_allowed()
Check if command *AssignResources* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

AssignResources(*argin*)
Assign resources to this subarray

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** (*list of str*) – the resources to be assigned

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_ReleaseResources_allowed()
Check if command *ReleaseResources* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

ReleaseResources(*argin*)
Delta removal of assigned resources.

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** (*list of str*) – the resources to be released

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_ReleaseAllResources_allowed()
Check if command *ReleaseAllResources* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

ReleaseAllResources()
Remove all resources to tear down to an empty subarray.

To modify behaviour for this command, modify the do() method of the command class.

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_Configure_allowed()
Check if command *Configure* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Configure(*argin*)
Configures the capabilities of this subarray

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** (*string*) – configuration specification

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_Scan_allowed()
Check if command *Scan* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Scan(*argin*)
Start scanning

To modify behaviour for this command, modify the do() method of the command class.

Parameters **argin** (*Array of str*) – Information about the scan

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_EndScan_allowed()
Check if command *EndScan* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

EndScan()
End the scan

To modify behaviour for this command, modify the do() method of the command class.

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_End_allowed()
Check if command *End* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

End()
End the scan block.

To modify behaviour for this command, modify the do() method of the command class.

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_Abort_allowed()
Check if command *Abort* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Abort()
Abort any long-running command such as `Configure()` or `Scan()`.

To modify behaviour for this command, modify the do() method of the command class.

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_ObsReset_allowed()
Check if command *ObsReset* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

ObsReset()
Reset the current observation process.

To modify behaviour for this command, modify the do() method of the command class.

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_Restart_allowed()
Check if command *Restart* is allowed in the current device state.

Raises **tango.DevFailed** – if the command is not allowed

Returns `True` if the command is allowed

Return type boolean

Restart()
Restart the subarray. That is, deconfigure and release all resources.

To modify behaviour for this command, modify the do() method of the command class.

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

Return type (*ResultCode*, str)

8.2 Device State Model

class ska_tango_base.**SKASubarrayStateModel**(*logger*, *op_state_callback=None*, *admin_mode_callback=None*, *obs_state_callback=None*)

Implements the state model for the SKASubarray

Initialises the model. Note that this does not imply moving to INIT state. The INIT state is managed by the model itself.

Parameters

- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this state model.
- **op_state_callback** (*callable*) – A callback to be called when a transition implies a change to op state
- **admin_mode_callback** (*callable*) – A callback to be called when a transition causes a change to device admin_mode
- **obs_state_callback** (*callable*) – A callback to be called when a transition causes a change to device obs_state

8.3 Resource Manager

class ska_tango_base.**SKASubarrayResourceManager**

A simple class for managing subarray resources

Constructor for SKASubarrayResourceManager

assign(*resources*)

Assign some resources

Todo Currently implemented for testing purposes to take a JSON string encoding a dictionary with key 'example'. In future this will take a collection of resources.

Parameters **resources** (*JSON string*) – JSON-encoding of a dictionary, with resources to assign under key 'example'

release(*resources*)

Release some resources

Todo Currently implemented for testing purposes to take a JSON string encoding a dictionary with key 'example'. In future this will take a collection of resources.

Parameters **resources** (*JSON string*) – JSON-encoding of a dictionary, with resources to assign under key 'example'

release_all()

Release all resources

get()

Get current resources

Returns a set of current resources.

Return type set of string

CHAPTER

NINE

CSP SUB-ELEMENT MASTER

This module implements a general Master device for a CSP Sub-element. CspSubElementMaster

Master device for SKA CSP Subelement.

9.1 Tango Device Class

class ska_tango_base.**CspSubElementMaster**(**args: Any*, ***kwargs: Any*)

Master device for SKA CSP Subelement.

Properties:

- **Device Property**

  PowerDelayStandbyOn

  - Delay in sec between power-up stages in Standby<-> On transitions.
  - Type:'DevFloat'

  PowerDelayStandByOff

  - Delay in sec between power-up stages in Standby-> Off transition.
  - Type:'DevFloat'

PowerDelayStandbyOn = tango.server.device_property(dtype=DevFloat, default_value=2.0)

PowerDelayStandbyOff = tango.server.device_property(dtype=DevFloat, default_value=1.5)

powerDelayStandbyOn = tango.server.attribute(dtype=DevFloat, access=tango.AttrWriteTyp

Device attribute.

powerDelayStandbyOff = tango.server.attribute(dtype=DevFloat, access=tango.AttrWriteTy

Device attribute.

onProgress = tango.server.attribute(dtype=DevUShort, label=onProgress, max_value=100, m

Device attribute.

onMaximumDuration = tango.server.attribute(dtype=DevFloat, access=tango.AttrWriteType.

Device attribute.

onMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=onMeasuredDuration, t

Device attribute.

standbyProgress = tango.server.attribute(dtype=DevUShort, label=standbyProgress, max_va

Device attribute.

standbyMaximumDuration = tango.server.attribute(dtype=DevFloat, access=tango.AttrWrite
Device attribute.

standbyMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=standbyMeasured
Device attribute.

offProgress = tango.server.attribute(dtype=DevUShort, label=offProgress, max_value=100
Device attribute.

offMaximumDuration = tango.server.attribute(dtype=DevFloat, access=tango.AttrWriteType
Device attribute.

offMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=offMeasuredDuration
Device attribute.

totalOutputDataRateToSdp = tango.server.attribute(dtype=DevFloat, label=totalOutputDat
Device attribute.

loadFirmwareProgress = tango.server.attribute(dtype=DevUShort, label=loadFirmwareProgr
Device attribute.

loadFirmwareMaximumDuration = tango.server.attribute(dtype=DevFloat, access=tango.Attr
Device attribute.

loadFirmwareMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=loadFirmwa
Device attribute.

init_command_objects()
Sets up the command objects

class InitCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementMaster's init_device() "command".

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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()
Method always executed before any TANGO command is executed.

delete_device()
Hook to delete resources allocated in init_device.

This method allows for any memory or other resources allocated in the init_device method to be released. This method is called by the device destructor and by the device Init command.

read_powerDelayStandbyOn()
Return the powerDelayStandbyOn attribute.

write_powerDelayStandbyOn(*value*)
Set the powerDelayStandbyOn attribute.

read_onProgress()
Return the onProgress attribute.

read_onMaximumDuration()
Return the onMaximumDuration attribute.

write_onMaximumDuration(*value*)
Set the onMaximumDuration attribute.

read_onMeasuredDuration()
Return the onMeasuredDuration attribute.

read_standbyProgress()
Return the standbyProgress attribute.

read_standbyMaximumDuration()
Return the standbyMaximumDuration attribute.

write_standbyMaximumDuration(*value*)
Set the standbyMaximumDuration attribute.

read_standbyMeasuredDuration()
Return the standbyMeasuredDuration attribute.

read_offProgress()
Return the offProgress attribute.

read_offMaximumDuration()
Return the offMaximumDuration attribute.

write_offMaximumDuration(*value*)
Set the offMaximumDuration attribute.

read_offMeasuredDuration()
Return the offMeasuredDuration attribute.

read_totalOutputDataRateToSdp()
Return the totalOutputDataRateToSdp attribute.

read_powerDelayStandbyOff()
Return the powerDelayStandbyOff attribute.

write_powerDelayStandbyOff(*value*)
Set the powerDelayStandbyOff attribute.

read_loadFirmwareProgress()
Return the loadFirmwareProgress attribute.

read_loadFirmwareMaximumDuration()
Return the loadFirmwareMaximumDuration attribute.

write_loadFirmwareMaximumDuration(*value*)
Set the loadFirmwareMaximumDuration attribute.

read_loadFirmwareMeasuredDuration()
Return the loadFirmwareMeasuredDuration attribute.

class LoadFirmwareCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementMaster's LoadFirmware command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for device LoadFirmware() command.

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

Return type (*ResultCode*, str)

check_allowed()

Check if the command is in the proper state (State/adminMode) to be executed. The master device has to be in OFF/MAINTENACE to process the LoadFirmware command.

Raises `CommandError` if command not allowed

Returns `True` if the command is allowed.

Return type boolean

class PowerOnDevicesCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementMaster's PowerOnDevices command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for device PowerOnDevices() command.

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

Return type (*ResultCode*, str)

check_allowed()

Check if the command is in the proper state to be executed. The master device has to be in ON to process the PowerOnDevices command.

: raises: `CommandError` if command not allowed : return: `True` if the command is allowed. : rtype: boolean

class PowerOffDevicesCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementMaster's PowerOffDevices command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for device PowerOffDevices() command.

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

Return type (*ResultCode*, str)

check_allowed()

Check if the command is in the proper state to be executed. The master device has to be in ON to process the PowerOffDevices command.

: raises: `CommandError` if command not allowed : return: `True` if the command is allowed. : rtype: boolean

class ReInitDevicesCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementMaster's ReInitDevices command.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

Stateless hook for device ReInitDevices() command.

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

Return type (*ResultCode*, str)

check_allowed()

Check if the command is in the proper state to be executed. The master device has to be in ON to process the ReInitDevices command.

: raises: `CommandError` if command not allowed : return: `True` if the command is allowed. : rtype: boolean

is_LoadFirmware_allowed()

Check if the LoadFirmware command is allowed in the current state.

Raises `CommandError` if command not allowed

Returns `True` if command is allowed

Return type boolean

LoadFirmware(*argin*)
Deploy new versions of software and firmware and trigger a restart so that a Component initializes using a newly deployed version.

Parameters **argin** (*'DevVarStringArray'*) – A list of three strings: - The file name or a pointer to the filename specified as URL. - the list of components that use software or firmware package (file), - checksum or signing Ex: [‘file://firmware.txt’,’test/dev/1, test/dev/2, test/dev/3’, ‘918698a7fea3fa9da5996db001d33628’]

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_PowerOnDevices_allowed()
Check if the PowerOnDevice command is allowed in the current state.

:raises `tango.DevFailed` if command not allowed :return `True` if command is allowed :rtype: boolean

PowerOnDevices(*argin*)
Power-on a selected list of devices.

Parameters **argin** (*'DevVarStringArray'*) – List of devices (FQDNs) to power-on.

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_PowerOffDevices_allowed()
Check if the PowerOffDevices command is allowed in the current state.

Raises `tango.DevFailed` if command not allowed

Returns `True` if command is allowed

Return type boolean

PowerOffDevices(*argin*)
Power-off a selected list of devices.

Parameters **argin** (*'DevVarStringArray'*) – List of devices (FQDNs) to power-off.

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_ReInitDevices_allowed()
Check if the ReInitDevices command is allowed in the current state.

Raises `tango.DevFailed` if command not allowed

Returns `True` if command is allowed

Return type boolean

ReInitDevices(*argin*)
Reinitialize the devices passed in the input argument. The exact functionality may vary for different devices and sub-systems, each TANGO Device/Server should define what does ReInitDevices means. Ex: ReInitDevices FPGA -> reset ReInitDevices Master -> Restart ReInitDevices Leaf PC -> reboot

Parameters **argin** (*'DevVarStringArray'*) – List of devices (FQDNs) to re-initialize.

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

Return type (*ResultCode*, str)

CHAPTER

TEN

CSP SUB-ELEMENT OBSDEVICE

CspSubElementObsDevice

General observing device for SKA CSP Subelement.

10.1 Tango Device Class

class ska_tango_base.**CspSubElementObsDevice**(**args: Any*, ***kwargs: Any*)

General observing device for SKA CSP Subelement.

Properties:

- **Device Property**

  DeviceID

  - Identification number of the observing device.
  - Type:'DevUShort'

DeviceID = tango.server.device_property(dtype=DevUShort, default_value=1)

scanID = tango.server.attribute(dtype=DevULong64, label=scanID, doc=The scan identifica

Device attribute.

configurationID = tango.server.attribute(dtype=DevString, label=configurationID, doc=Th

Device attribute.

deviceID = tango.server.attribute(dtype=DevUShort, label=deviceID, doc=The observing de

Device attribute.

lastScanConfiguration = tango.server.attribute(dtype=DevString, label=lastScanConfigura

Device attribute.

sdpDestinationAddresses = tango.server.attribute(dtype=DevString, label=sdpDestinationA

Device attribute.

sdpLinkCapacity = tango.server.attribute(dtype=DevFloat, label=sdpLinkCapacity, doc=The

Device attribute.

sdpLinkActive = tango.server.attribute(dtype=('DevBoolean',), max_dim_x=100, label=sdpL

Device attribute.

healthFailureMessage = tango.server.attribute(dtype=DevString, label=healthFailureMessa

Device attribute.

init_command_objects()

Sets up the command objects

class InitCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementObsDevice's init_device() "command".

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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()
Method always executed before any TANGO command is executed.

delete_device()
Hook to delete resources allocated in init_device.

This method allows for any memory or other resources allocated in the init_device method to be released. This method is called by the device destructor and by the device Init command.

read_scanID()
Return the scanID attribute.

read_configurationID()
Return the configurationID attribute.

read_deviceID()
Return the deviceID attribute.

read_lastScanConfiguration()
Return the lastScanConfiguration attribute.

read_sdpDestinationAddresses()
Return the sdpDestinationAddresses attribute.

read_sdpLinkCapacity()
Return the sdpLinkCapacity attribute.

read_sdpLinkActive()
Return the sdpLinkActive attribute.

read_healthFailureMessage()
Return the healthFailureMessage attribute.

class ConfigureScanCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementObsDevices's ConfigureScan command.

Constructor for ConfigureScanCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (CspSubElementObsStateModel) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

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*)

Validate the configuration parameters against allowed values, as needed.

Parameters **argin** (*'DevString'*) – The JSON formatted string with configuration for the device.

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

Return type (*ResultCode*, str)

class ScanCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementObsDevices's Scan command.

Constructor for ScanCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (CspSubElementObsStateModel) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)

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)

validate_input(*argin*)

Validate the command input argument.

Parameters **argin** (*string*) – 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)

class EndScanCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementObsDevices's EndScan command.

Constructor for EndScanCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (CspSubElementObsStateModel) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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 GoToIdleCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementObsDevices's GoToIdle command.

Constructor for GoToIdle Command.

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (CspSubElementObsStateModel) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()

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 ObsResetCommand(*target*, *state_model*, *logger=None*)

A class for the CspSubElementObsDevices's ObsReset command.

Constructor for ObsReset Command.

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (CspSubElementObsStateModel) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

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(*target*, *state_model*, *logger=None*)
A class for the CspSubElementObsDevices's Abort command.

Constructor for Abort Command.

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (CspSubElementObsStateModel) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

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)

ConfigureScan(*argin*)
Configure the observing device parameters for the current scan.

Parameters **argin** ('DevString') – JSON formatted string with the scan configuration.

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(*argin*)
Start an observing scan.

Parameters **argin** ('DevString') – A string with 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)

EndScan()
End a running scan.

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

Return type (*ResultCode*, str)

GoToIdle()
Transit the device from READY to IDLE obsState.

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

Return type (*ResultCode*, str)

ObsReset()
Reset the observing device from a FAULT/ABORTED obsState to IDLE.

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()
Abort the current observing process and move the device to ABORTED obsState.

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

Return type (*ResultCode*, str)

10.2 Instance attributes

Here it is reported the list of the *instance attributes*.

- `scan_id`: the identification number of the scan. The scan ID is passed as argument of the *Scan* command. The attribute value is reported via TANGO attribute *scanID*.
- `_sdp_addresses`: a python dictionary with the SDP destination addresses for the output products. Depending on the sub-element (CBF, PSS, PST) this attribute can specify more than one destination address, as for example in CBF sub-element. The SDP destination addresses are specified at configuration. An SDP address specifies the MAC address, IP address and port of the endpoint. Below an example of how SDP addresses are specified in a Mid CBF configuration:

```
{
  ...
  "outputHost": [[0, "192.168.0.1"], [8184, "192.168.0.2"]],
  "outputMac": [[0, "06-00-00-00-00-01"]],
  "outputPort": [[0, 9000, 1], [8184, 9000, 1]]
  ...
}
```

  The value of this attribute is reported via the TANGO *sdpDestionationAddresses* attribute.

  Note: Not all the Sub-element observing devices are connected to the SDP (for example Mid VCCs).

- `_sdp_links_active`: a python list of boolean. Each list element reports the network connectivity of the corresponding link to SDP.
- `_sdp_links_capacity`: this attribute records the capacity in GB/s of the SDP link.
- `_config_id`: it stores the unique identificator associated to a JSON scan configuration. The value of this attribute is reported via the TANGO attriute *configID*.
- `_last_scan_configuration`: this attribute stores the last configuration successully programmed. The value is reported via the TANGO attribute *lastScanConfiguration*.
- `_health_failure_msg`: The value is reported via the TANGO attribute *healthFailureMesssage*.

CHAPTER

ELEVEN

CSP SUB-ELEMENT SUBARRAY

This module implements a generic Subarray device for a CSP Sub-element. The scope of this module is to provide a uniform access to a CSP Sub-element subarray from the CSP.LMC side. CspSubElementSubarray

Subarray device for SKA CSP SubElement

11.1 Tango Device Class

class ska_tango_base.**CspSubElementSubarray**(**args: Any*, ***kwargs: Any*)

Subarray device for SKA CSP SubElement

scanID = tango.server.attribute(dtype=DevULong64, label=scanID, doc=The scan identific

Device attribute.

configurationID = tango.server.attribute(dtype=DevString, label=configurationID, doc=T

Device attribute.

sdpDestinationAddresses = tango.server.attribute(dtype=DevString, access=tango.AttrWri

Device attribute.

outputDataRateToSdp = tango.server.attribute(dtype=DevFloat, label=outputDataRateToSdp

Device attribute.

lastScanConfiguration = tango.server.attribute(dtype=DevString, label=lastScanConfigur

Device attribute.

sdpLinkActive = tango.server.attribute(dtype=('DevBoolean',), max_dim_x=100, label=sdp

Device attribute.

listOfDevicesCompletedTasks = tango.server.attribute(dtype=DevString, label=listOfDevi

Device attribute.

configureScanMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=configure

Device attribute.

configureScanTimeoutExpiredFlag = tango.server.attribute(dtype=DevBoolean, label=confi

Device attribute.

assignResourcesMaximumDuration = tango.server.attribute(dtype=DevFloat, access=tango.A

Device attribute.

assignResourcesMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=assignR

Device attribute.

assignResourcesProgress = tango.server.attribute(dtype=DevUShort, label=assignResource

Device attribute.

assignResourcesTimeoutExpiredFlag = tango.server.attribute(dtype=DevBoolean, label=ass
Device attribute.

releaseResourcesMaximumDuration = tango.server.attribute(dtype=DevFloat, access=tango.
Device attribute.

releaseResourcesMeasuredDuration = tango.server.attribute(dtype=DevFloat, label=releas
Device attribute.

releaseResourcesProgress = tango.server.attribute(dtype=DevUShort, label=releaseResour
Device attribute.

releaseResourcesTimeoutExpiredFlag = tango.server.attribute(dtype=DevBoolean, label=ti
Device attribute.

init_command_objects()
Sets up the command objects

class InitCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementObsDevice's init_device() "command".

Create a new InitCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the SKASubarray device for which this class implements the command
- **state_model** (*DeviceStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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()
Method always executed before any TANGO command is executed.

delete_device()
Hook to delete resources allocated in init_device.

This method allows for any memory or other resources allocated in the init_device method to be released. This method is called by the device destructor and by the device Init command.

read_scanID()
Return the scanID attribute.

read_configurationID()
Return the configurationID attribute.

read_sdpDestinationAddresses()
Return the sdpDestinationAddresses attribute.

write_sdpDestinationAddresses(*value*)
Set the sdpDestinationAddresses attribute.

read_outputDataRateToSdp()
Return the outputDataRateToSdp attribute.

read_lastScanConfiguration()
Return the lastScanConfiguration attribute.

read_configureScanMeasuredDuration()
Return the configureScanMeasuredDuration attribute.

read_configureScanTimeoutExpiredFlag()
Return the configureScanTimeoutExpiredFlag attribute.

read_listOfDevicesCompletedTasks()
Return the listOfDevicesCompletedTasks attribute.

read_assignResourcesMaximumDuration()
Return the assignResourcesMaximumDuration attribute.

write_assignResourcesMaximumDuration(*value*)
Set the assignResourcesMaximumDuration attribute.

read_assignResourcesMeasuredDuration()
Return the assignResourcesMeasuredDuration attribute.

read_assignResourcesProgress()
Return the assignResourcesProgress attribute.

read_assignResourcesTimeoutExpiredFlag()
Return the assignResourcesTimeoutExpiredFlag attribute.

read_releaseResourcesMaximumDuration()
Return the releaseResourcesMaximumDuration attribute.

write_releaseResourcesMaximumDuration(*value*)
Set the releaseResourcesMaximumDuration attribute.

read_releaseResourcesMeasuredDuration()
Return the releaseResourcesMeasuredDuration attribute.

read_releaseResourcesProgress()
Return the releaseResourcesProgress attribute.

read_releaseResourcesTimeoutExpiredFlag()
Return the releaseResourcesTimeoutExpiredFlag attribute.

read_sdpLinkActive()
Return the sdpLinkActive attribute.

class ConfigureScanCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementObsDevices's ConfigureScan command.

Constructor for ConfigureScanCommand

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin*)
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)

validate_input(*argin*)
Validate the configuration parameters against allowed values, as needed. :param argin: The JSON formatted string with configuration for the device. :type argin: 'DevString' :return: A tuple containing a return code and a string message. :rtype: (ResultCode, str)

class GoToIdleCommand(*target*, *state_model*, *logger=None*)
A class for the CspSubElementObsDevices's GoToIdle command.

Constructor for GoToIdle Command.

Parameters

- **target** (*object*) – the object that this command acts upon; for example, the CspSubElementObsDevice device for which this class implements the command
- **state_model** (*SKASubarrayStateModel*) – the state model that this command uses to check that it is allowed to run, and that it drives with actions.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do()
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)

ConfigureScan(*argin*)
Configure a complete scan for the subarray.

Parameters **argin** ('*DevString*') – JSON formatted string with the scan configuration.

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(*argin*)
Redirect to ConfigureScan method. Configure a complete scan for the subarray.

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

GoToIdle()
Transit the subarray from READY to IDLE obsState.

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

End()
Transit the subarray from READY to IDLE obsState. Redirect to GoToIdle command.

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

CHAPTER

TWELVE

SKA CONTROL MODEL

Module for SKA Control Model (SCM) related code.

For further details see the SKA1 CONTROL SYSTEM GUIDELINES (CS_GUIDELINES MAIN VOLUME) Document number: 000-000000-010 GDL And architectural updates: https://jira.skatelescope.org/browse/ADR-8 https://confluence.skatelescope.org/pages/viewpage.action?pageId=105416556

The enumerated types mapping to the states and modes are included here, as well as other useful enumerations.

class ska_tango_base.control_model.**HealthState**(*value*)

Python enumerated type for healthState attribute.

OK = 0

TANGO Device reports this state when ready for use, or when entity adminMode is NOT_FITTED or RESERVED.

The rationale for reporting health as OK when an entity is NOT_FITTED or RESERVED is to ensure that it does not pop-up unnecessarily on drill-down fault displays with healthState UNKNOWN, DEGRADED or FAILED while it is expected to not be available.

DEGRADED = 1

TANGO Device reports this state when only part of functionality is available. This value is optional and shall be implemented only where it is useful.

For example, a subarray may report healthState as DEGRADED if one of the dishes that belongs to a subarray is unresponsive, or may report healthState as FAILED.

Difference between DEGRADED and FAILED health shall be clearly identified (quantified) and documented. For example, the difference between DEGRADED and FAILED subarray can be defined as the number or percent of the dishes available, the number or percent of the baselines available, sensitivity, or some other criterion. More than one criteria may be defined for a TANGO Device.

FAILED = 2

TANGO Device reports this state when unable to perform core functionality and produce valid output.

UNKNOWN = 3

Initial state when health state of entity could not yet be determined.

class ska_tango_base.control_model.**AdminMode**(*value*)

Python enumerated type for adminMode attribute.

ONLINE = 0

SKA operations declared that the entity can be used for observing (or other function it implements). During normal operations Elements and subarrays (and all other entities) shall be in this mode. TANGO Devices that implement adminMode as read-only attribute shall always report adminMode=ONLINE. adminMode=ONLINE is also used to indicate active Subarrays or Capabilities.

OFFLINE = 1
SKA operations declared that the entity is not used for observing or other function it provides. A subset of the monitor and control functionality may be supported in this mode. `adminMode=OFFLINE` is also used to indicate unused Subarrays and unused Capabilities. TANGO devices report `state=DISABLED` when `adminMode=OFFLINE`.

MAINTENANCE = 2
SKA operations declared that the entity is reserved for maintenance and cannot be part of scientific observations, but can be used for observing in a 'Maintenance Subarray'.

`MAINTENANCE` mode has different meaning for different entities, depending on the context and functionality. Some entities may implement different behaviour when in `MAINTENANCE` mode.

For each TANGO Device, the difference in behaviour and functionality in `MAINTENANCE` mode shall be documented. `MAINTENANCE` is the factory default for `adminMode`. Transition out of `adminMode=NOT_FITTED` is always via `MAINTENANCE`; an engineer/operator has to verify that the entity is operational as expected before it is set to `ONLINE` (or `OFFLINE`).

NOT_FITTED = 3
SKA operations declared the entity as `NOT_FITTED` (and therefore cannot be used for observing or other function it provides). TM shall not send commands or queries to the Element (entity) while in this mode.

TANGO devices shall report `state=DISABLE` when `adminMode=NOT_FITTED`; higher level entities (Element, Sub-element, component, Subarray and/or Capability) which 'use' `NOT_FITTED` equipment shall report operational `state` as `DISABLE`. If only a subset of higher-level functionality is affected, overall `state` of the higher-level entity that uses `NOT_FITTED` equipment may be reported as `ON`, but with `healthState=DEGRADED`. Additional queries may be necessary to identify which functionality and capabilities are available.

Higher-level entities shall intelligently exclude `NOT_FITTED` items from `healthState` and Element Alerts/Telescope Alarms; e.g. if a receiver band in DSH is `NOT_FITTED` and there is no communication to that receiver band, then DSH shall not raise Element Alerts for that entity and it should not report `healthState=FAILED` because of an entity that is `NOT_FITTED`.

RESERVED = 4
This mode is used to identify additional equipment that is ready to take over when the operational equipment fails. This equipment does not take part in the operations at this point in time. TANGO devices report `state=DISABLED` when `adminMode=RESERVED`.

class ska_tango_base.control_model.**ObsState**(*value*)
Python enumerated type for `obsState` attribute - the observing state.

EMPTY = 0
The sub-array is ready to observe, but is in an undefined configuration and has no resources allocated.

RESOURCING = 1
The system is allocating resources to, or deallocating resources from, the subarray. This may be a complete de/allocation, or it may be incremental. In both cases it is a transient state and will automatically transition to IDLE when complete. For some subsystems this may be a very brief state if resourcing is a quick activity.

IDLE = 2
The subarray has resources allocated and is ready to be used for observing. In normal science operations these will be the resources required for the upcoming SBI execution.

CONFIGURING = 3
The subarray is being configured ready to scan. On entry to the state no assumptions can be made about the previous conditions. It is a transient state and will automatically transition to READY when it completes normally.

READY = 4
The subarray is fully prepared to scan, but is not actually taking data or moving in the observed coordinate system (it may be tracking, but not moving relative to the coordinate system).

SCANNING = 5
The subarray is taking data and, if needed, all components are synchronously moving in the observed coordinate system. Any changes to the sub-systems are happening automatically (this allows for a scan to cover the case where the phase centre is moved in a pre-defined pattern).

ABORTING = 6
The subarray is trying to abort what it was doing due to having been interrupted by the controller.

ABORTED = 7
The subarray has had its previous state interrupted by the controller, and is now in an aborted state.

RESETTING = 8
The subarray device is resetting to the IDLE state.

FAULT = 9
The subarray has detected an error in its observing state making it impossible to remain in the previous state.

RESTARTING = 10
The subarray device is restarting, as the last known stable state is where no resources were allocated and the configuration undefined.

class ska_tango_base.control_model.**ObsMode**(*value*)
Python enumerated type for obsMode attribute - the observing mode.

IDLE = 0
The obsMode shall be reported as IDLE when obsState is IDLE; else, it will correctly report the appropriate value. More than one observing mode can be active in the same subarray at the same time.

IMAGING = 1
Imaging observation is active.

PULSAR_SEARCH = 2
Pulsar search observation is active.

PULSAR_TIMING = 3
Pulsar timing observation is active.

DYNAMIC_SPECTRUM = 4
Dynamic spectrum observation is active.

TRANSIENT_SEARCH = 5
Transient search observation is active.

VLBI = 6
Very long baseline interferometry observation is active.

CALIBRATION = 7
Calibration observation is active.

class ska_tango_base.control_model.**ControlMode**(*value*)
Python enumerated type for controlMode attribute.

REMOTE = 0
TANGO Device accepts commands from all clients.

LOCAL = 1
TANGO Device accepts only from a 'local' client and ignores commands and queries received from TM or any other 'remote' clients. This is typically activated by a switch, or a connection on the local control

interface. The intention is to support early integration of DISHes and stations. The equipment has to be put back in `REMOTE` before clients can take control again. `controlMode` may be removed from the SCM if unused/not needed.

Note: Setting *controlMode* to *LOCAL* **is not a safety feature**, but rather a usability feature. Safety has to be implemented separately to the control paths.

class ska_tango_base.control_model.**SimulationMode**(*value*)

Python enumerated type for `simulationMode` attribute.

FALSE = 0

A real entity is connected to the control system.

TRUE = 1

A simulator is connected to the control system, or the real entity acts as a simulator.

class ska_tango_base.control_model.**TestMode**(*value*)

Python enumerated type for `testMode` attribute.

This enumeration may be replaced and extended in derived classes to add additional custom test modes. That would require overriding the base class `testMode` attribute definition.

NONE = 0

Normal mode of operation. No test mode active.

TEST = 1

Element (entity) behaviour and/or set of commands differ for the normal operating mode. To be implemented only by devices that implement one or more test modes. The Element documentation shall provide detailed description.

class ska_tango_base.control_model.**LoggingLevel**(*value*)

Python enumerated type for `loggingLevel` attribute.

OFF = 0

FATAL = 1

ERROR = 2

WARNING = 3

INFO = 4

DEBUG = 5

CHAPTER

THIRTEEN

COMMANDS

This module provides abstract base classes for device commands, and a ResultCode enum.

class ska_tango_base.commands.**ResultCode**(*value*)
Python enumerated type for command return codes.

OK = 0
The command was executed successfully.

STARTED = 1
The command has been accepted and will start immediately.

QUEUED = 2
The command has been accepted and will be executed at a future time

FAILED = 3
The command could not be executed.

UNKNOWN = 4
The status of the command is not known.

class ska_tango_base.commands.**BaseCommand**(*target*, *state_model*, *logger=None*)
Abstract base class for Tango device server commands. Ensures the command is run, and that if the command errors, the "fatal_error" action will be called on the state model.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

do(*argin=None*)
Hook for the functionality that the command implements. This class provides stub functionality; subclasses should subclass this method with their command functionality.

Parameters **argin** (*ANY*) – the argument passed to the Tango command, if present

fatal_error()
Callback for a fatal error in the command, such as an unhandled exception.

class ska_tango_base.commands.**ResponseCommand**(*target*, *state_model*, *logger=None*)

Abstract base class for a tango command handler, for commands that execute a procedure/operation and return a (ResultCode, message) tuple.

Creates a new BaseCommand object for a device.

Parameters

- **state_model** (*SKABaseClassStateModel or a subclass of same*) – the state model that this command uses, for example to raise a fatal error if the command errors out.
- **target** (*object*) – the object that this base command acts upon. For example, the device that this BaseCommand implements the command for.
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

class ska_tango_base.commands.**ActionCommand**(*target*, *state_model*, *action_hook*, *start_action=False*, *logger=None*)

Abstract base class for a tango command, which checks a state model to find out whether the command is allowed to be run, and after running, sends an action to that state model, thus driving device state.

Create a new ActionCommand for a device.

Parameters

- **target** (*object*) – the object that this base command acts upon. For example, the device that this ActionCommand implements the command for.
- **action_hook** (*string*) – a hook for the command, used to build actions that will be sent to the state model; for example, if the hook is "scan", then success of the command will result in action "scan_succeeded" being sent to the state model.
- **start_action** (*boolean*) – whether the state model supports a start action (i.e. to put the state model into an transient state while the command is running); default False
- **logger** (*a logger that implements the standard library logger interface*) – the logger to be used by this Command. If not provided, then a default module logger will be used.

check_allowed()

Checks whether the command is allowed to be run in the current state of the state model.

Returns True if the command is allowed to be run

Raises **StateModelError** – if the command is not allowed to be run

is_allowed()

Whether this command is allowed to run in the current state of the state model.

Returns whether this command is allowed to run

Return type boolean

started()

Action to perform upon starting the comand.

succeeded()

Callback for the successful completion of the command.

failed()

Callback for the failed completion of the command.

CHAPTER

FOURTEEN

STATE MACHINE

The state machine module implements three fundamental SKA state machines:

- the admin mode state machine
- the operational state (opState, represented in TANGO devices by TANGO state) state machine
- the observation state machine.

14.1 Admin mode state machine

The admin mode state machine allows for transitions between the five administrative modes:

- NOT_FITTED: this is the lowest state of readiness, representing devices that cannot be deployed without some external action, such as plugging hardware in or updating network settings.)
- RESERVED: the device is fitted but redundant to other devices. It is ready to take over should other devices fail.
- OFFLINE: the device has been declared by SKA operations not currently to be used for operations (or whatever other function it provides)
- MAINTENANCE: the device cannot be used for science purposes but can be operationed for engineering / maintenance purposes, such as testing, debugging, etc
- ONLINE: the device can be used for science purposes.

The admin mode state machine allows for

- any transition between the modes NOT_FITTED, RESERVED and OFFLINE (e.g. an unfitted device being fitted as a redundant or non-redundant device, a redundant device taking over when another device fails, etc)
- any transition between the modes OFFLINE, MAINTENANCE and ONLINE (e.g. an online device being taken offline or put into maintenance mode to diagnose a fault, a faulty device moving between maintenance and offline mode as it undergoes sporadic periods of diagnosis.

Diagrams of the admin mode state machine are shown below.

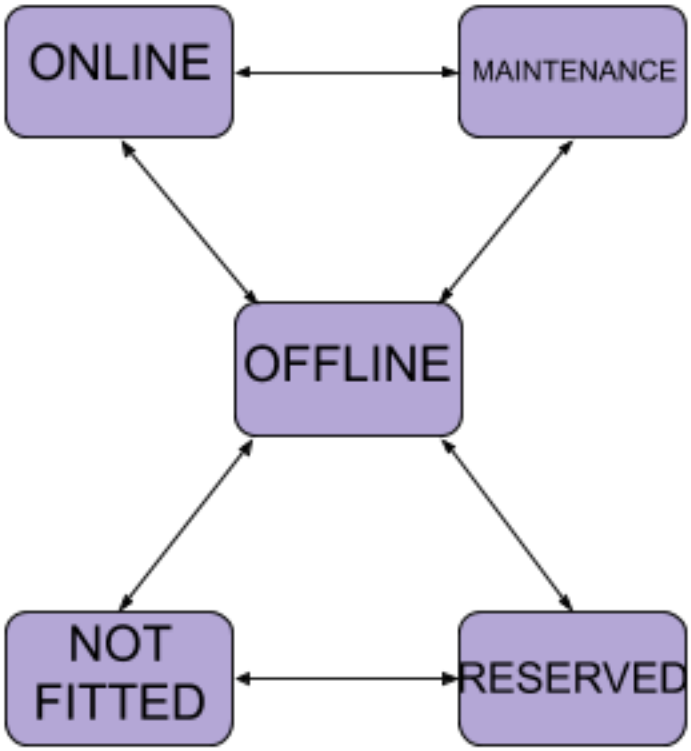

Fig. 1: Diagram of the admin mode state machine, as designed

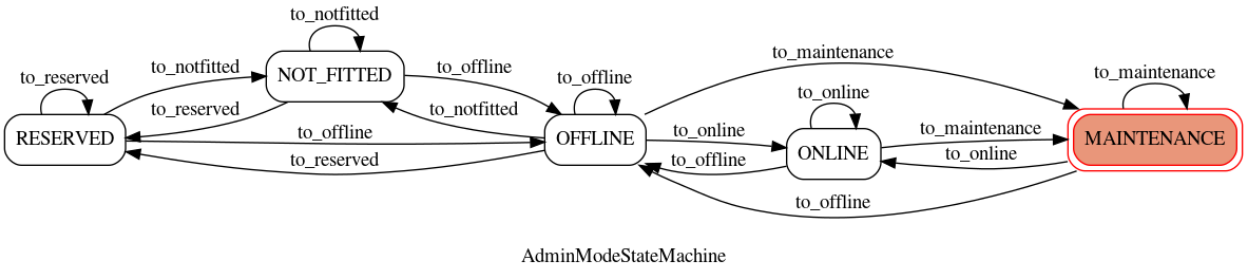

Fig. 2: Diagram of the admin mode state machine, automatically generated from the implementation. The equivalence of this diagram to the diagram above demonstrates that the machine has been implemented as designed.

14.2 Operational state machine

The operational state (opState) machine represents the operational state of a SKA device. It is represented in TANGO devices using the TANGO "state", so the states used are a subset of the TANGO states: INIT, FAULT, DISABLE, STANDBY, OFF and ON.

- INIT: the device is currently initialising
- FAULT: the device has experienced an error from which it could not recover.
- DISABLE: the device is in its lowest state of readiness, from which it may take some time to become fully operational. For example, if the device manages hardware, that hardware may be switched off.
- STANDBY: the device is unready, but can be made ready quickly. For example, if the device manages hardware, that hardware may be in a low-power standby mode.
- OFF: the device is fully operational but is not currently in use
- ON: the device is in use

The operational state state machine allows for:

- transition from INIT or FAULT into any of the three "readiness states" DISABLE, STANDBY and OFF.
- all transitions between these three "readiness states" DISABLE, STANDBY and OFF.
- transition between OFF and ON.

Unfortunately, operational state is inextricably coupled with admin mode: there are admin modes that imply disablement, and operational states such as ON should not be possible in such admin modes.

To facilitate this, the entire operational state state machine is accessible only when the admin mode is ONLINE or MAINTENANCE. When in any other admin mode, the only permitted operational states are INIT, FAULT and DISABLE. This constraint is implemented into the operational state state machine by

- three extra states: INIT_ADMIN, FAULT_ADMIN and DISABLED_ADMIN
- two extra transition triggers: "admin_on" and "admin_off", which allow for transition between INIT and INIT_ADMIN; FAULT and FAULT_ADMIN; and DISABLE and DISABLE_ADMIN.

This implementation minimises the coupling between admin mode and operational state, allowing the two machines to be conceptualised almost separately.

Diagrams of the operational state state machine are shown below.

14.3 Observation state machine

The observation state machine is implemented by devices that manage observations (currently only subarray devices).

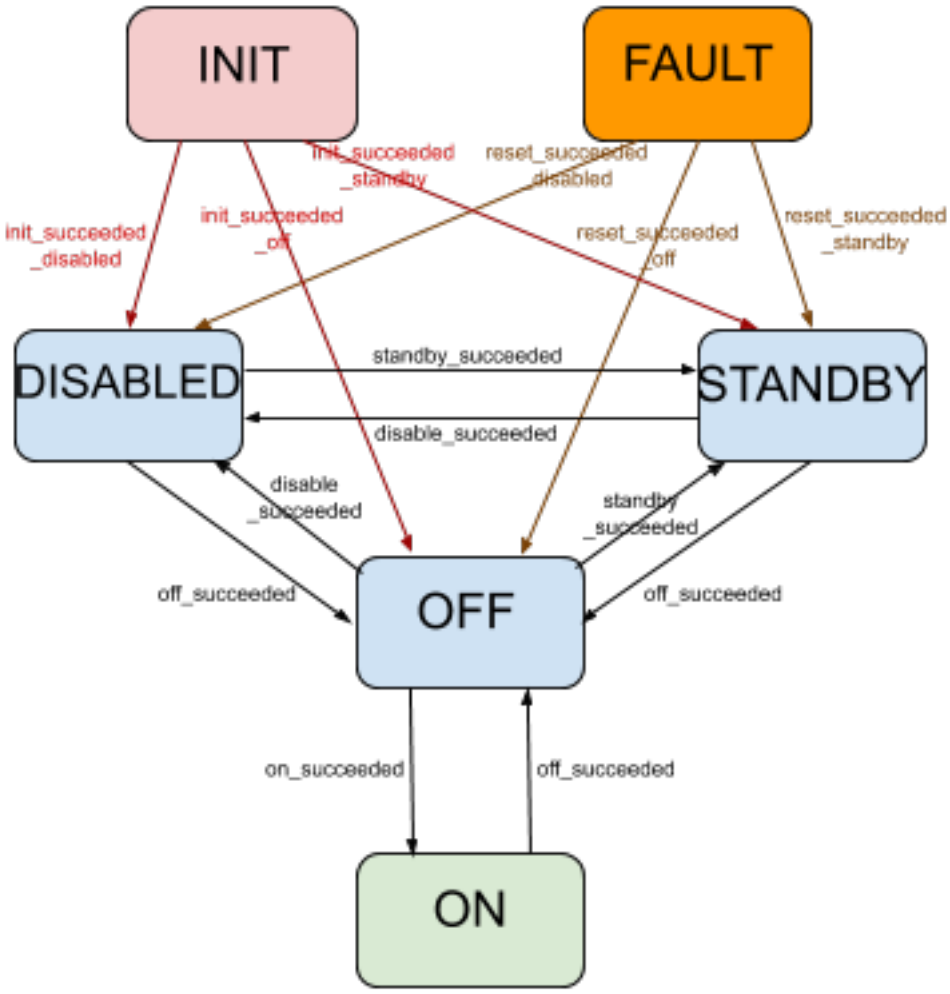

Fig. 3: Diagram of the operational state (opState) state machine, as designed, ignoring coupling with admin mode

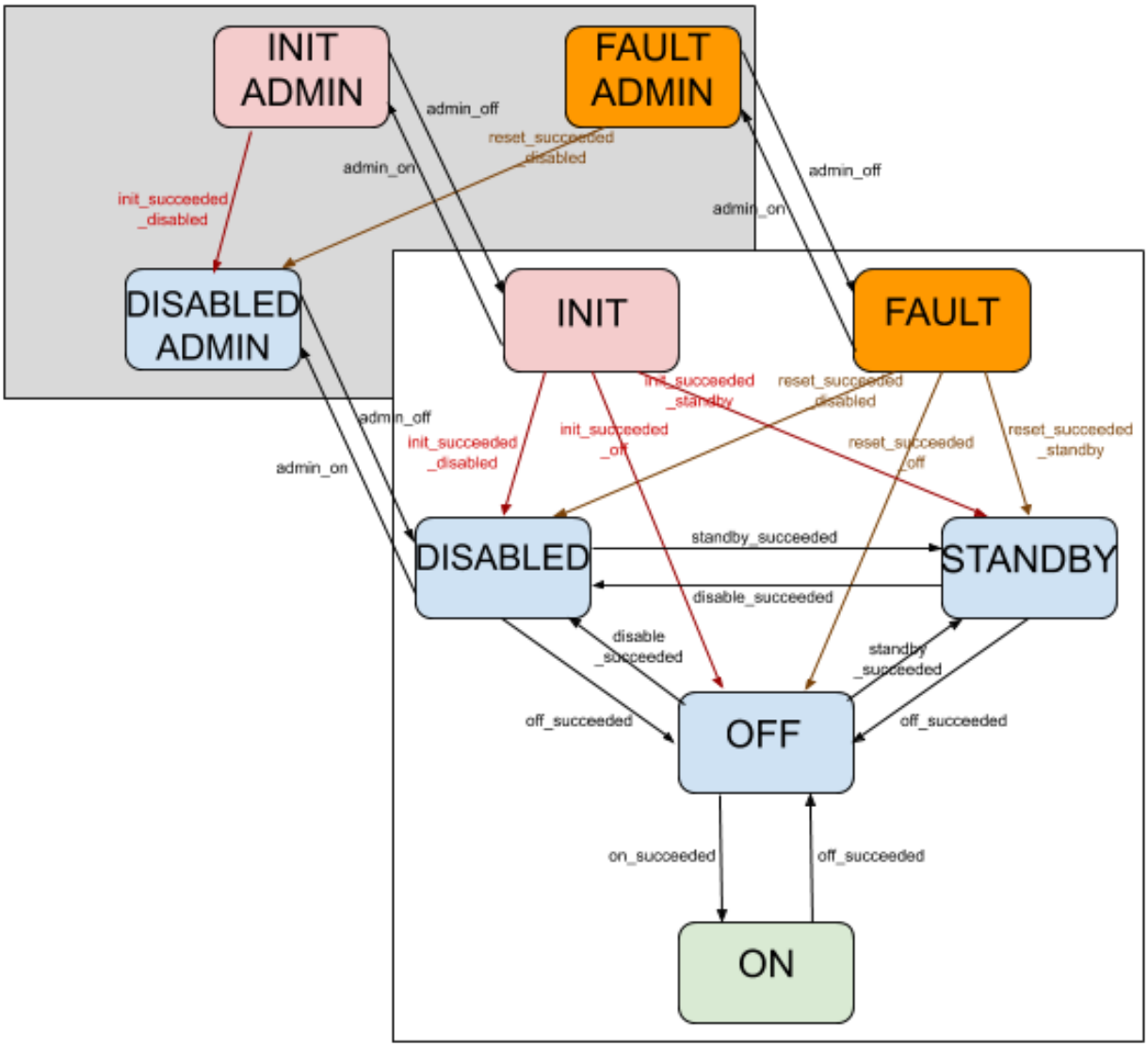

Fig. 4: Diagram of the operational state (opState) state machine, as designed, showing coupling with admin mode

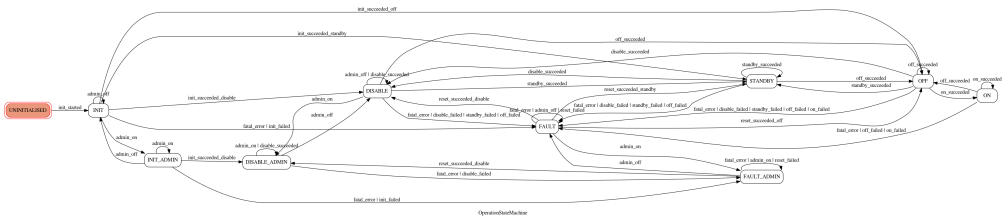

Fig. 5: Diagram of the operational state state machine, automatically generated from the implementation. The equivalence of this diagram to the diagram above demonstrates that the machine has been implemented as designed.

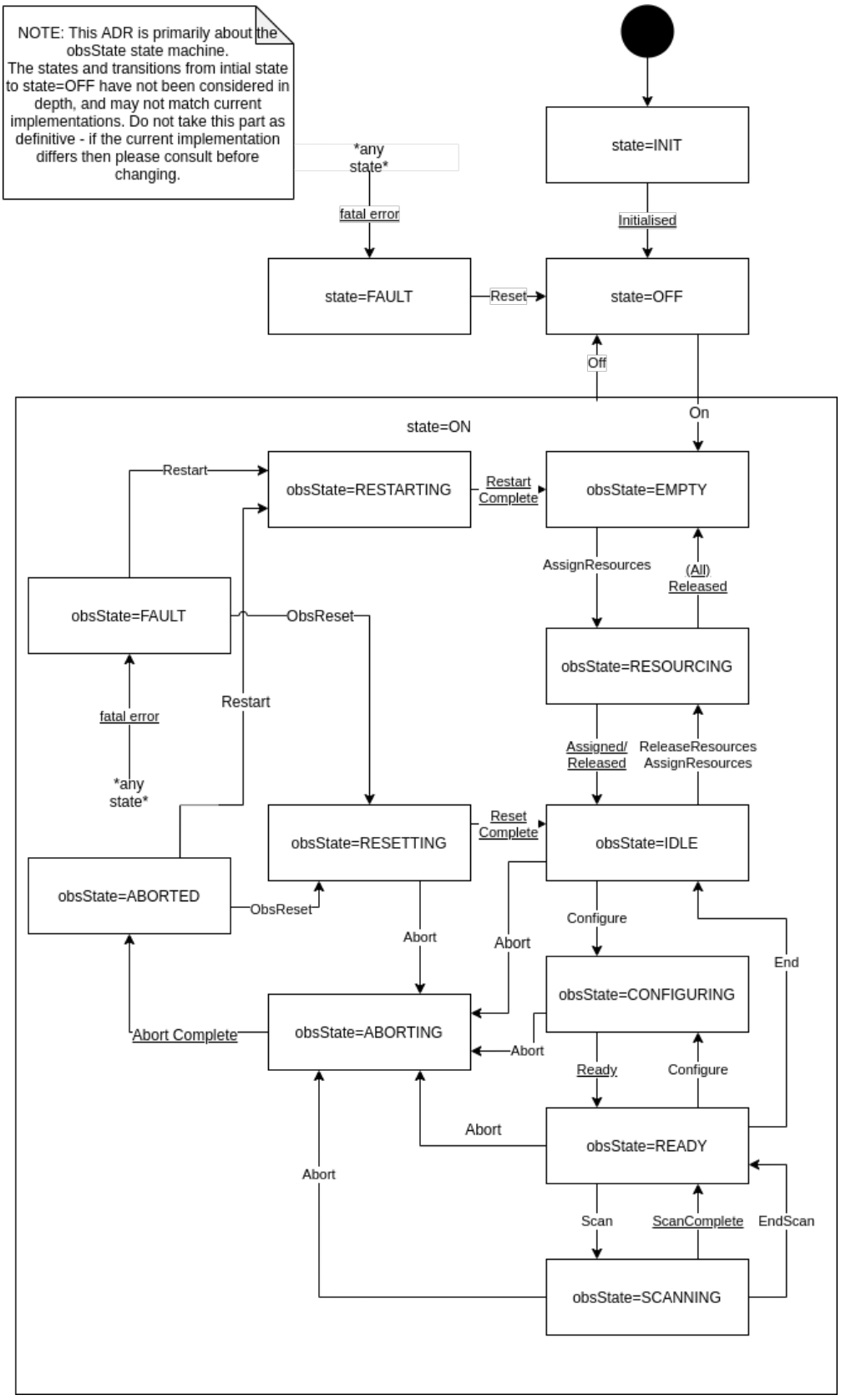

Fig. 6: Diagram of the observation state machine, as decided and published in ADR-8.

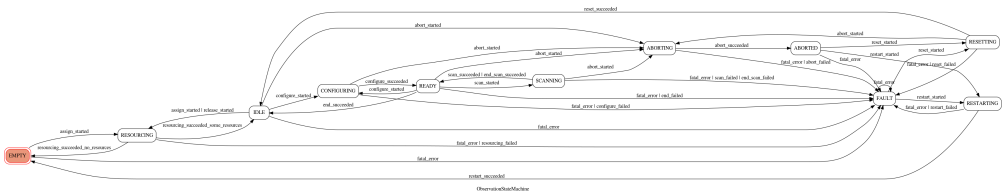

Fig. 7: Diagram of the observation state machine, automatically generated from the implementation. The equivalance of this diagram to the diagram previous demonstrates that the machine has been implemented in conformance with ADR-8.

14.4 CSP SubElement ObsDevice state machine

This state machine is implemented for the CSP SubElement devices, different from the subarrays, that manage observations.

Compared to the SKA Observation State Machine, it implements a smaller number of states, number that can be further descreased depending on the necessities of the different sub-elements.

The implemented states for the current state machine are:

- IDLE: this is the observing state after the device intialization.
- CONFIGURING: transitional state to report the device configuration is in progress. *Need to understand if this state is really required by the observing devices of any CSP sub-element.*
- READY: the device is configured and is ready to perform observations
- SCANNING: the device is performing the observation.
- ABORTING: the device is processing an abort. Need to understand if this state is really required by the observing devices of any CSP sub-element.
- ABORTED: the device has completed the abort request.
- FAULT: the device has experienced an error from which it can be recovered only via manual intervention invoking a reset command that force the device to the base state (IDLE).

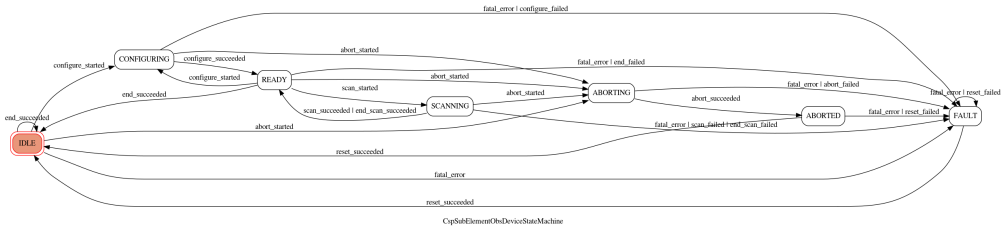

Fig. 8: Diagram of the CSP SubElement observation state machine, automatically generated from the implementation.

14.5 API

This module contains specifications of SKA state machines.

class ska_tango_base.state_machine.**OperationStateMachine**(**args: Any*, ***kwargs: Any*)

State machine for operational state ("opState").

The states supported are "UNINITIALISED", "INIT", "FAULT", "DISABLE", "STANDBY", "OFF" and "ON".

The states "INIT", "FAULT" and "DISABLE" also have "INIT_ADMIN", "FAULT_ADMIN" and "DISABLE_ADMIN" flavours to represent these states in situations where the device being modelled has been administratively disabled.

Initialises the state model.

Parameters

- **callback** (*callable*) – A callback to be called when a transition implies a change to op state
- **extra_kwargs** – Additional keywords arguments to pass to super class initialiser (useful for graphing)

class ska_tango_base.state_machine.**AdminModeStateMachine**(**args: Any*, ***kwargs: Any*)

The state machine governing admin modes

Initialises the admin mode state machine model.

Parameters

- **callback** (*callable*) – A callback to be called whenever there is a transition to a new admin mode value
- **extra_kwargs** – Additional keywords arguments to pass to super class initialiser (useful for graphing)

class ska_tango_base.state_machine.**ObservationStateMachine**(**args: Any*, ***kwargs: Any*)

The observation state machine used by an observing subarray, per ADR-8.

Initialises the model.

Parameters

- **callback** (*callable*) – A callback to be called when the state changes
- **extra_kwargs** – Additional keywords arguments to pass to super class initialiser (useful for graphing)

This module contains specifications of the CSP SubElement Observing state machine.

class ska_tango_base.csp_subelement_state_machine.**CspSubElementObsDeviceStateMachine**(**args: Any*, ***kwarg Any*)

The observation state machine used by a generic CSP Sub-element ObsDevice (derived from SKAObsDevice).

Initialises the model.

Parameters

- **callback** (*callable*) – A callback to be called when the state changes

- **extra_kwargs** – Additional keywords arguments to pass to super class initialiser (useful for graphing)

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

C

D

L

Q

R

S

T

U

V

W