ska-control-model

Release 0.3.1

Drew Devereux <drew.devereux@csiro.au>

Jun 08, 2023

CONTENTS:

CHAPTER

ONE

ADMIN MODE

class ska_control_model.**AdminMode**(*value*)

Python enumerated type for device admin mode.

An admin mode represents user intent as to how the component under control will be used.

MAINTENANCE = 2

The component under control can be used for maintainance purposes, such as testing, debugging or commissioning, as part of a "maintenance subarray".. It may not be used for normal operations.

While in this mode, the control system actively monitors and controls its component, but may only support a subset of normal functionality. Alarms and alerts will usually be suppressed.

`MAINTENANCE` mode has different meaning for different components, 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.

NOT_FITTED = 3

The component cannot be used for any purposes because it is not fitted; for example, faulty equipment has been removed and not yet replaced, leaving nothing *in situ* to monitor.

While in this mode, the control system reports state `DISABLED`. All monitoring and control functionality is disabled because there is no component to monitor.

OFFLINE = 1

The component under control shall not be monitored or controlled by the control system.

Either the component under control shall not be used at all, or it is under external control (such as the local control of a field technician).

While in this mode, the control system reports its state as `DISABLE`. Since monitoring of the component is not occurring, the control system does not issue alarms, alerts and other events.

ONLINE = 0

The component under control can be used for normal operations, such as observing. While in this mode, the control system actively monitors and controls the component under control.

Control system elements that implement admin mode as a read-only attribute shall always report the admin mode to be `ONLINE`.

RESERVED = 4

The component is fitted, but only for redundancy purposes. It is additional equipment that does not take part in operations at this time, but is ready to take over when the operational equipment fails.

While in this mode, the control system reports state `DISABLED`. All monitoring and control functionality is disabled.

class ska_control_model.**AdminModeModel**(*logger*, *callback=None*, *state_machine_factory=<class 'ska_control_model.admin_mode._AdminModeMachine'>*)

This class implements the state model for admin mode.

The model supports the five admin modes defined by the values of the *AdminMode* enum. It 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.)

The actions supported are:

- **to_not_fitted**
- **to_reserved**
- **to_offline**
- **to_maintenance**
- **to_online**

A diagram of the admin mode model, as designed, is shown below

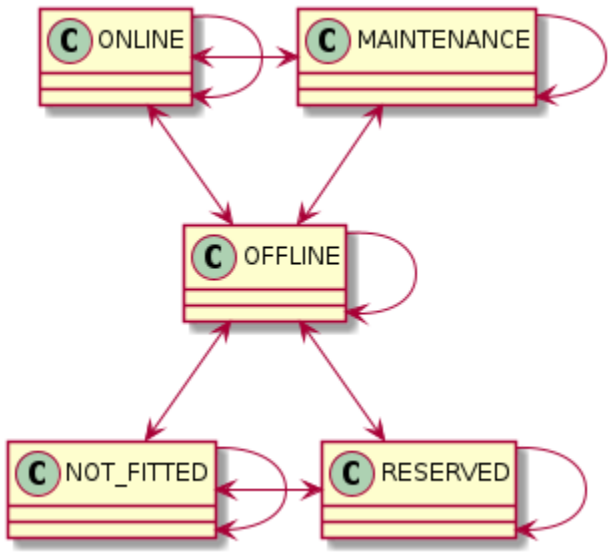

Fig. 1: Diagram of the admin mode model

__init__(*logger*, *callback=None*, *state_machine_factory=<class 'ska_control_model.admin_mode._AdminModeMachine'>*)

Initialise the state model.

Parameters

- **logger** (Logger) – the logger to be used by this state model.
- **callback** (Optional[Callable[[*AdminMode*], None]]) – A callback to be called when the state machine for admin_mode reports a change of state

- **state_machine_factory** (Callable) – a callable that returns a state machine for this model to use

__weakref__

list of weak references to the object (if defined)

property admin_mode: *AdminMode*

Return the admin_mode.

Return type
AdminMode

Returns
admin_mode of this state model

is_action_allowed(*action*, *raise_if_disallowed=False*)

Return whether a given action is allowed in the current state.

Parameters

- **action** (str) – an action, as given in the transitions table
- **raise_if_disallowed** (bool) – whether to raise an exception if the action is disallowed, or merely return False (optional, defaults to False)

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

Return type
bool

Returns
whether the action is allowed in the current state

perform_action(*action*)

Perform an action on the state model.

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

Return type
None

CHAPTER

TWO

OPERATING STATE

class ska_control_model.**OpStateModel**(*logger*, *callback=None*, *state_machine_factory=None*)

This class implements the state model for operational state ("opState").

The model supports the following states, represented as values of the `tango.DevState` enum.

- **INIT**: the control system is initialising.
- **DISABLE**: the control system has been told not to monitor the system under control.
- **UNKNOWN**: the control system is monitoring (or at least trying to monitor) the system under control, but is unable to determine its state.
- **OFF**: the control system is monitoring the system under control, which is powered off.
- **STANDBY**: the control system is monitoring the system under control, which is in low-power standby mode.
- **ON**: the control system is monitoring the system under control, which is turned on.
- **FAULT**: the control system is monitoring the system under control, which has failed or is in an inconsistent state.

The actions supported are:

- **init_invoked**: the control system has started initialising.
- **init_completed**: the control system has finished initialising.
- **component_disconnected**: the control system his disconnected from the system under control (for example because admin mode was set to OFFLINE). Note, this action indicates a deliberate, control-system-initiated, disconnect; a lost connection would be indicated by a "component_unknown" action.
- **component_unknown**: the control system is unable to determine the state of the system under control.
- **component_off**: the system under control has been switched off
- **component_standby**: the system under control has switched to low-power standby mode
- **component_on**: the system under control has been switched on.
- **component_fault**: the system under control has experienced a fault.
- **component_no_fault**: the system under control has stopped experiencing a fault.

A diagram of the operational state model, as implemented, is shown below.

The following hierarchical diagram is more explanatory; however note that the implementation does *not* use a hierarchical state machine.

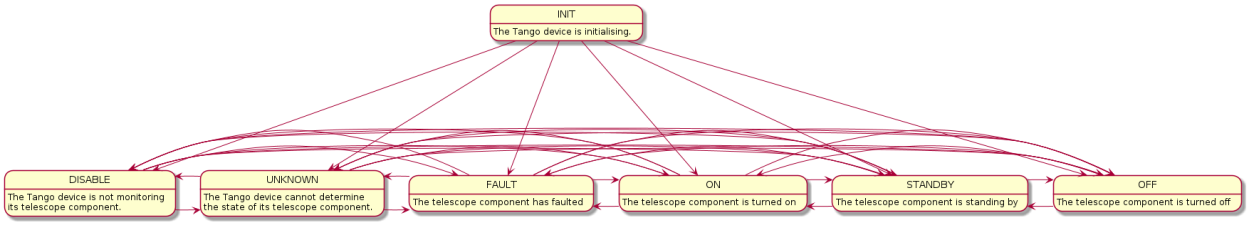

Fig. 1: Diagram of the operational state model

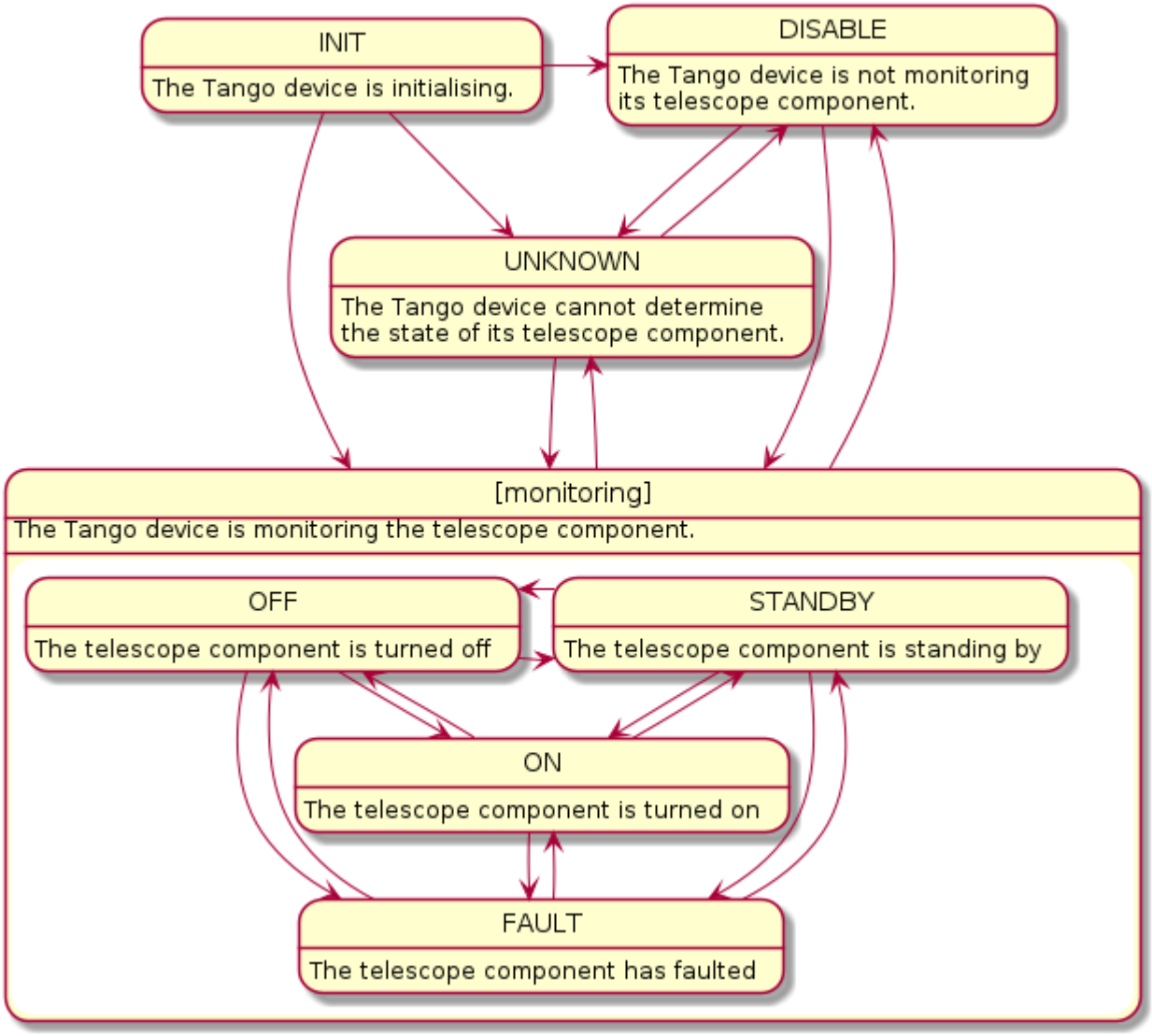

Fig. 2: Diagram of the operational state model

__init__(*logger*, *callback=None*, *state_machine_factory=None*)

Initialise the operational state model.

Parameters

- **logger** (Logger) – the logger to be used by this state model.
- **callback** (Optional[Callable]) – A callback to be called when the state machine for op_state reports a change of state
- **state_machine_factory** (Optional[Callable]) – a callable that returns a state machine for this model to use

__weakref__

list of weak references to the object (if defined)

is_action_allowed(*action*, *raise_if_disallowed=False*)

Return whether a given action is allowed in the current state.

Parameters

- **action** (str) – an action, as given in the transitions table
- **raise_if_disallowed** (bool) – whether to raise an exception if the action is disallowed, or merely return False (optional, defaults to False)

Raises

StateModelError – if the action is unknown to the state machine

Return type

bool

Returns

whether the action is allowed in the current state

property op_state: tango.DevState

Return the op state.

Return type

DevState

Returns

the op state of this state model

perform_action(*action*)

Perform an action on the state model.

Parameters

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

Return type

None

CHAPTER

THREE

OBSERVING STATE

class ska_control_model.**ObsState**(*value*)

Python enumerated type for observing state.

ABORTED = 7

The subarray is in an aborted state.

ABORTING = 6

The subarray has been interrupted and is aborting what it was doing.

CONFIGURING = 3

The subarray is being configured for an observation.

This is a transient state; the subarray will automatically transition to READY when configuring completes normally.

EMPTY = 0

The sub-array has no resources allocated and is unconfigured.

FAULT = 9

The subarray has detected an error in its observing state.

IDLE = 2

The subarray has resources allocated but is unconfigured.

READY = 4

The subarray is fully prepared to scan, but is not scanning.

It may be tracked, but it is not moving in the observed coordinate system, nor is it taking data.

RESETTING = 8

The subarray device is resetting to a base (EMPTY or IDLE) state.

RESOURCING = 1

Resources are being allocated to, or deallocated from, the subarray.

In normal science operations these will be the resources required for the upcoming SBI execution.

This may be a complete de/allocation, or it may be incremental. In both cases it is a transient state; when the resourcing operation completes, the subarray will automatically transition to EMPTY or IDLE, according to whether the subarray ended up having resources or not.

For some subsystems this may be a very brief state if resourcing is a quick activity.

RESTARTING = 10

The subarray device is restarting.

After restarting, the subarray will return to EMPTY state, with no allocated resources and no configuration defined.

SCANNING = 5

The subarray is scanning.

It 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).

class ska_control_model.**ObsStateModel**(*logger*, *callback=None*, *state_machine_factory=<class 'ska_control_model.obs_state._ObsStateMachine'>*)

Implements the observation state model for subarray.

The model supports all of the states of the *ObsState* enum:

- **EMPTY**: the subarray is unresourced
- **RESOURCING**: the subarray is performing a resourcing operation
- **IDLE**: the subarray is resourced but unconfigured
- **CONFIGURING**: the subarray is performing a configuring operation
- **READY**: the subarray is resourced and configured
- **SCANNING**: the subarray is scanning
- **ABORTING**: the subarray is aborting
- **ABORTED**: the subarray has aborted
- **RESETTING**: the subarray is resetting from an ABORTED or FAULT state back to IDLE
- **RESTARTING**: the subarray is restarting from an ABORTED or FAULT state back to EMPTY
- **FAULT**: the subarray has encountered a observation fault.

A diagram of the subarray observation state model is shown below. This model is non-deterministic as diagrammed, but the underlying state machines has extra states and transitions that render it deterministic. This class simply maps those extra classes onto valid ObsState values.

__init__(*logger*, *callback=None*, *state_machine_factory=<class 'ska_control_model.obs_state._ObsStateMachine'>*)

Initialise the model.

Parameters

- **logger** (Logger) – the logger to be used by this state model.
- **callback** (Optional[Callable]) – A callback to be called when a transition causes a change to device obs_state
- **state_machine_factory** (Callable) – a callable that returns a state machine for this model to use

__weakref__

list of weak references to the object (if defined)

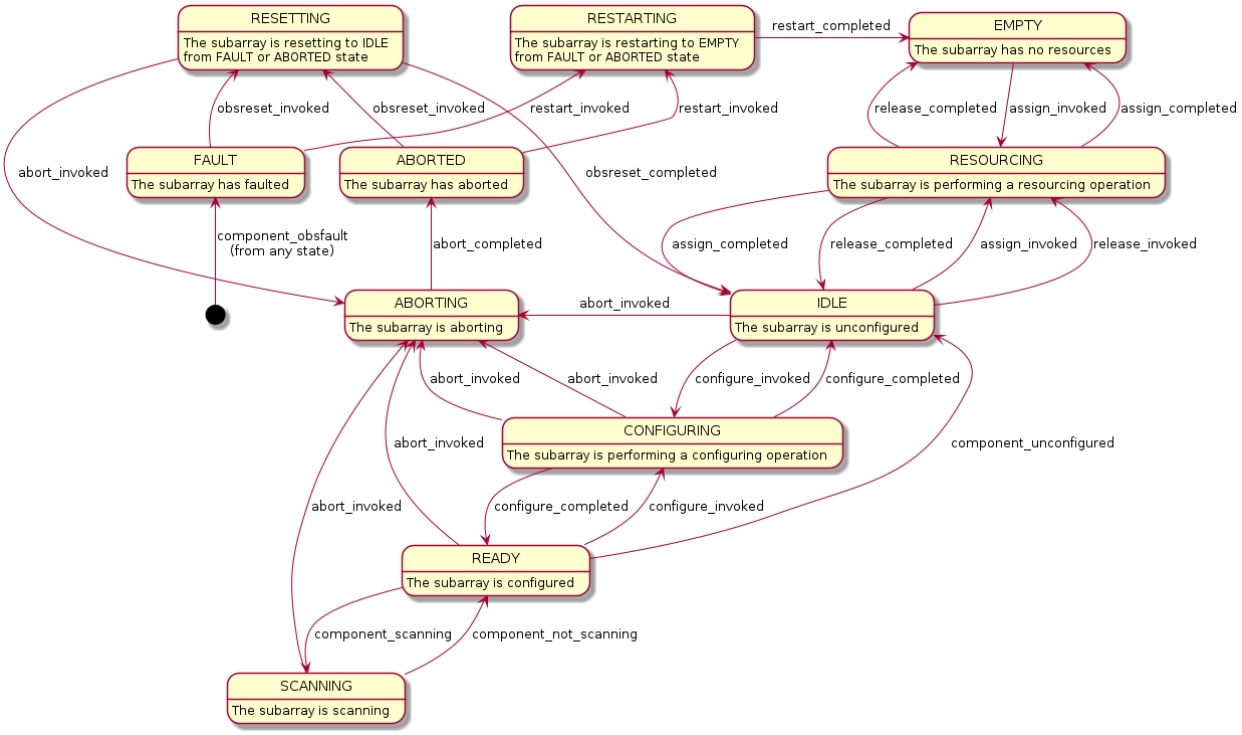

Fig. 1: Diagram of the subarray observation state model

is_action_allowed(*action*, *raise_if_disallowed=False*)

Return whether a given action is allowed in the current state.

Parameters

- **action** (str) – an action, as given in the transitions table
- **raise_if_disallowed** (bool) – whether to raise an exception if the action is disallowed, or merely return False (optional, defaults to False)

Raises

StateModelError – if the action is unknown to the state machine

Return type

bool

Returns

whether the action is allowed in the current state

property obs_state: Optional[*ObsState*]

Return the obs_state.

Return type

Optional[*ObsState*]

Returns

obs_state of this state model

perform_action(*action*)

Perform an action on the state model.

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

Return type
None

CHAPTER

FOUR

OBSERVING MODE

class ska_control_model.**ObsMode**(*value*)

Python enumerated type for observing mode.

CALIBRATION = 7

Calibration observation is active.

DYNAMIC_SPECTRUM = 4

Dynamic spectrum observation is active.

IDLE = 0

The observing mode shall be IDLE when the observing state is IDLE (see *ObsState*). Otherwise, 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.

TRANSIENT_SEARCH = 5

Transient search observation is active.

VLBI = 6

Very long baseline interferometry observation is active.

CHAPTER

FIVE

HEALTH STATE

class ska_control_model.**HealthState**(*value*)

Python enumerated type for health state.

DEGRADED = 1

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

For example, a subarray may report its health state as DEGRADED if one of the dishes that belongs to a subarray is unresponsive (or may report health state as FAILED).

Difference between DEGRADED and FAILED health state shall be clearly identified (quantified) and documented. For example, the difference between a DEGRADED and FAILED subarray might 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 criterion may be defined for a device.

FAILED = 2

The device reports this state when unable to perform core functionality and produce valid output.

OK = 0

A device reports this state when there are no failures that are assessed as affecting the ability of the device to perform its function.

UNKNOWN = 3

The device reports this state when unable to determine its health.

This is also an initial state, indicating that health state has not yet been determined.

CHAPTER

SIX

SIMULATION MODE

class ska_control_model.**SimulationMode**(*value*)

Python enumerated type for simulation mode.

FALSE = 0

The control system is connected to a real entity.

TRUE = 1

The control system is connected to a simulator.

It may be connected to an entirely separate simulator, or the real entity may be itself in a simulation mode.

CHAPTER

SEVEN

CONTROL MODE

class ska_control_model.**ControlMode**(*value*)

Python enumerated type for control mode.

LOCAL = 1

Monitoring and control operations are accepted only from a "local" client.

Commands and queries received from TM or any other "remote" clients are ignored.

This mode 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.

Note: LOCAL control mode is **not a safety feature**, but rather a usability feature. Safety must be implemented separately from the control paths.

REMOTE = 0

Monitoring and control operations are accepted from all clients.

CHAPTER

EIGHT

TEST MODE

class ska_control_model.**TestMode**(*value*)

Python enumerated type for test mode.

NONE = 0

Normal mode of operation. No test mode active.

TEST = 1

Test mode active.

The element's behaviour and/or interface differs from the normal operating mode.

To be implemented only by devices that implement one or more test modes. The element documentation shall provide detailed description.

CHAPTER

NINE

COMMUNICATION STATUS

class ska_control_model.**CommunicationStatus**(*value*)

The status of communication with the system under controk.

DISABLED = 0

Communication is disabled.

The control system is not trying to establish/maintain a channel of communication with the system under control. For example:

- if communication with the system under control is connection-oriented, then there is no connection, and the control system is not trying to establish a connection.
- if communication is by event subscription, then the control system is unsubscribed from events.
- if communication is by polling, then the control system is not performing that polling.

ESTABLISHED = 2

The control system has established a channel of communication with the system under control. For example:

- if communication with the system under control is connection-oriented, then the control system has connected to the system under control.
- if communication is by polling, then the control system is polling the system under control, and the system under control is responsive.

NOT_ESTABLISHED = 1

Communication is sought but not established.

The control system is trying to establish/maintain a channel of communication with the system under control, but that channel is not currently established. For example:

- if communication with the system under control is connection-oriented, then the control system has not yet succeeded in establishing the connection, or the connection has been broken.

CHAPTER

TEN

POWER STATE

class ska_control_model.**PowerState**(*value*)

Enumerated type for power state.

Used by components that rely upon a power supply, such as hardware.

NO_SUPPLY = 1

The component is unsupplied with power and cannot be commanded on.

For example, the power mode of a TPM will be NO_SUPPLY if the subrack that powers the TPM is turned off: not only is the TPM off, but it cannot even be turned on (until the subrack has been turned on).

OFF = 2

The component is turned off but can be commanded on.

ON = 4

The component is powered on and running in fully-operational mode.

STANDBY = 3

The component is powered on and running in low-power standby mode.

UNKNOWN = 0

The power mode is not known.

CHAPTER

ELEVEN

LOGGING LEVEL

class ska_control_model.**LoggingLevel**(*value*)

Python enumerated type for logging level.

DEBUG = 5

Logs of information relevant only for debugging

ERROR = 2

Logs of errors.

FATAL = 1

Logs of critical events that result in component shutdown or failure.

INFO = 4

Logs of information relevant to users.

OFF = 0

Logging is turned off.

WARNING = 3

Logs of warnings.

CHAPTER

TWELVE

RESULT CODE

class ska_control_model.**ResultCode**(*value*)

Python enumerated type for command result codes.

ABORTED = 7

The command in progress has been aborted.

FAILED = 3

The command could not be executed.

NOT_ALLOWED = 6

The command is not allowed to be executed.

OK = 0

The command was executed successfully.

QUEUED = 2

The command has been accepted and will be executed at a future time.

REJECTED = 5

The command execution has been rejected.

STARTED = 1

The command has been accepted and will start immediately.

UNKNOWN = 4

The status of the command is not known.

CHAPTER

THIRTEEN

TASK STATUS

class ska_control_model.**TaskStatus**(*value*)

The status of a task.

A task is any operation that is being performed asynchronously.

ABORTED = 3

The task has been aborted.

COMPLETED = 5

The task was completed.

Note that this does not necessarily imply that the task was executed successfully. Whether the task succeeded or failed is a matter for the *ResultCode*. The COMPLETED value indicates only that execution of the task ran to completion.

FAILED = 7

The task failed to complete.

Note that this should not be used for a task that executes to completion, but does not achieve its goal. This kind of domain-specific notion of "succeeded" versus "failed" should be passed in a *ResultCode*. Here, FAILED means that the task executor has detected a failure of the task to run to completion. For example, execution of the task might have resulted in the raising of an uncaught exception.

IN_PROGRESS = 2

The task is being executed.

NOT_FOUND = 4

The task is not found.

QUEUED = 1

The task has been accepted and will be executed at a future time.

REJECTED = 6

The task was rejected.

STAGING = 0

The request to execute the task has not yet been acted upon.

CHAPTER

FOURTEEN

FAULTS

This module defines faults that are part of the SKA control model.

exception ska_control_model.faults.**StateModelError**

Error in state machine model related to transitions or state.

__weakref__

list of weak references to the object (if defined)

CHAPTER

FIFTEEN

UTILS

This module defines utils used in this package.

ska_control_model.utils.**for_testing_only**(*func*, *_testing_check*=<*function* <*lambda*>>)

Return a function that checks that it is being called in testing.

This is a decorator that only calls the decorated function after first checking that it is being called in testing. If called outside of testing, a warning is raised.

```
@for_testing_only
def _straight_to_state(self, state):
    ...
```

Parameters
func (Callable) – the function to be decorated

Return type
Callable

Returns
the decorated function

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

W