# Contents

1 What is tango-simlib?  
1.1 tango-simlib: Easily generate TANGO device simulators .......................... 3  
1.2 Introduction ......................................................................................... 3  
1.3 Basic Usage ......................................................................................... 4  
1.4 Device Simulators ............................................................................... 4  
1.5 tango_simlib modules ......................................................................... 7  
1.6 tango_simlib.utilities modules .......................................................... 8  
1.7 tango_simlib.examples modules ........................................................... 10  
1.8 License .............................................................................................. 14  
1.9 Contribute ......................................................................................... 14  
1.10 Contact us ....................................................................................... 14  

2 Indices and tables  

Python Module Index  

i
tango-simlib is a library that aids the data-driven development of TANGO device simulators. It aims to make it easy to develop basic simulators while making it possible to implement more complex simulators.
What is tango-simlib?

1.1 tango-simlib: Easily generate TANGO device simulators

1.2 Introduction

tango-simlib is a library that aids the data-driven development of TANGO device simulators. It aims to make it easy to develop basic simulators while making it possible to implement more complex simulators. In addition to the simulated device interface, a separate TANGO simulation-control interface is generated, allowing the simulator to be manipulated via a back-channel to simulate e.g. failure conditions on the simulated device interface.

Using only the basic TANGO interface description captured via a POGO generated XMI file, a basic simulator with randomly varying attributes and no-op command handlers can be generated with no further coding. Attribute simulation parameters and simple command behaviour can be specified using a Simulator Description Datafile (SimDD). The format of this file is currently a working proposal and subject to change. A more formal format specification is being worked on.

Note that tango-simlib does not generate simulator code. Rather, the simulator’s behaviour is driven by the description data at run-time using Python’s dynamic programming features. If the description files (XMI or SimDD) are modified, the simulator device server only needs to be restarted for the changes to take effect.

1.2.1 Rationale

During the development of the control and monitoring (CAM) systems for the KAT-7 (KAT-7-wiki) and MeerKAT (MeerKAT-wiki) at SKASA it was found that having control-interface simulators available for all hardware and sub-systems that CAM needs to control and monitor is an incredibly valuable resource. In early CAM development it:

- Makes it possible to start developing the CAM system before hardware or vendor-provided hardware simulators are available.
- Allows gaps in interfaces to be identified early on in the development process.

As development progresses, the full, actual, MeerKAT/KAT-7 CAM is run against the simulated devices, allowing CAM functionality to be tested without the need of real telescope hardware. The simulators also expose a back-channel that
can be used to modify the behaviour of the simulators during tests e.g. to simulate error conditions. This is exploited by CAM developers in their own development environments for day to day development tasks and also allows daily automated functional integration tests to be run. It was also found that initially having even a very simple simulator available is quite valuable, and that for many devices the simple simulator is always sufficient.

The KAT-7 and MeerKAT telescopes use the Karoo Array Telescope Control Protocol (KATCP) for inter-device, subsystem and component communications. In light of that a library was developed that makes it very easy to code a basic simulator, providing no-op command (KATCP request) handlers and randomly varying attribute (KATCP sensor) values along with the back-channel interface for “free”.

The planned SKA telescope project that the SKASA team is participating in has decided to standardise on the TANGO control systems framework. This library is an attempt to bring the same simulation approach used for the KAT-7 and MeerKAT telescope to the TANGO world.

1.3 Basic Usage

1.3.1 Installation

Note that installation requires the TANGO binary prerequisites to be installed. If you cannot install the PyTango package you will not be able to install tango-simlib.

Installation from source, working directory where source is checked out

```
$ pip install .
```

This package is available on PYPI, allowing

```
$ pip install tango-simlib
```

1.4 Device Simulators

1.4.1 Generating Simulators

The basic way of generating a TANGO device simulator using this library is to make use of the TANGO simulator generator module. Give it a path to the description files (XMI or SimDD or both).

```
$ tango-simlib-generator --sim-data-file Weather.xmi
  --dserver-name weather-DS
  --directory .
```

This will generate a python executable file in your current working directory named weather-DS.

In order to run this generated device simulator code, you can execute the tango-launcher script, a helper script which will register the TANGO device server, setup any required device properties and in turn start up the device server process, all in one go.

```
$ tango-simlib-launcher --name mkat_sim/weather/1 --class Weather
  --name mkat_simcontrol/weather/1
  --class WeatherSimControl
  --server-command weather-DS --port 0
  --server-instance tango-launched
  --key:mkat_sim/weather/1:model_ ...
```
1.4.2 Ready-made Simulators

Weather simulators

A code snippet of starting the Weather simulator generated from the Weather.xmi file with a SimControl instance using the tango_launcher.py script. Change directory to tango_simlib/examples/

```
$ tango-simlib-launcher --name mkat_sim/weather/1 --class Weather\
   --name mkat_simcontrol/weather/1\
   --class WeatherSimControl\
   --server-command weather1.py --port 0\
   --server-instance tango-launched\
   --put-device-property mkat_simcontrol/weather/1:model_key:mkat_sim/weather/1
```

An example of starting the Weather simulator generated from the Weather_SimDD.json file with a SimControl instance using the tango_launcher.py script.

```
$ tango-simlib-launcher --name mkat_sim/weather/2 --class Weather\
   --name mkat_simcontrol/weather/2\
   --class WeatherSimControl\
   --server-command weather2.py --port 0\
   --server-instance tango-launched\
   --put-device-property mkat_simcontrol/weather/2:model_key:mkat_sim/weather/2
```

**MeerKAT Video Display System simulator**

An example of starting the VDS simulator generated from both the MkatVds.xmi and the MkatVds_SimDD.json files with a SimControl instance using the tango_launcher.py script.

```
$ tango-simlib-launcher --name mkat_sim/vds/1 --class MkatVds\
   --name mkat_simcontrol/vds/1\
   --class MkatVdsSimControl\
   --server-command mkat_vds.py --port 0\
   --server-instance tango-launched\
   --put-device-property mkat_simcontrol/vds/1:model_key:mkat_sim/vds/1
```

Once the tango-simlib-tango-launcher script has been executed, the TANGO server will be created in the TANGO database. The TANGO device server will be registered along with its properties and the server process will be started. This will start the server instance which has the two classes Weather and WeatherSimControl registered under it, respectively, which in turn will start the devices from each of the TANGO classes.

1.4.3 Screenshots of Interfaces

This is what you would have in the TANGO DB once the device server has been registered

In this instance, we have the simulated device in an alarm state after executing the SetOffRainStorm command on the test device interface, or what we call the simulator controller.
Fig. 1: Figure 1. A snapshot of the TANGO DB viewed using JIVE - the TANGO-DB browser.

Fig. 2: Figure 2. A view of the sim device and its associated sim control interface using the TANGO Application ToolKit (ATK) client framework.
1.5 tango_simlib modules

1.5.1 tango_simlib.main module

1.5.2 tango_simlib.model module

1.5.3 tango_simlib.quantities module

```python
class tango_simlib.quantities.ConstantQuantity(start_value=None, start_time=None, meta=None)
    Bases: tango_simlib.quantities.Quantity
    A quantity that does not change unless explicitly set

    default_val(t)
    Set a default value of True to the quantity
    t [float] Time to update quantity

    next_val(t)
    Returns the last value as the next simulated value
    t [float] Time to update quantity

class tango_simlib.quantities.GaussianSlewLimited(mean, std_dev, max_slew_rate=inf, meta=None, min_bound=-inf, max_bound=inf, start_value=None, start_time=None)
    Bases: tango_simlib.quantities.Quantity
    A Gaussian random variable a slew-rate limit and clipping

    mean [float] Gaussian mean value
    std_dev [float] Gaussian standard deviation
    max_slew_rate [float] Maximum quantity slew rate in amount per second. Random values will be clipped to satisfy this condition.
    min_bound [float] Minimum quantity value, random values will be clipped if needed.
    max_bound [float] Maximum quantity value, random values will be clipped if needed.

    adjustable_attributes = frozenset(['last_update_time', 'max_slew_rate', 'last_val', 'mean'])

    next_val(t)
    Returns the next value of the simulation
    t [float] Time to update quantity

class tango_simlib.quantities.Quantity(start_value=None, start_time=None, meta=None)
    Bases: object
    Attributes that should be adjustable via a simulation control interface

    start_time [float] The initial time when a quantity is updated.
    start_value [float] The initial value of a quantity.
    meta [dict] This data structure must contain all the attribute description data of all quantities that represent tango device simulator attributes. List of all available tango attribute description data: abs_change, archive_abs_change, archive_period, archive_rel_change, label, max_alarm, max_value, max_warning, min_alarm, min_value, delta_t, delta_val, description, display_unit, format, min_warning, period,
```

1.5. tango_simlib modules
rel_change e.g. meta=dict(label="Outside Temperature", dtype=float) TODO (AR) 2016-07-27 : Ideally these properties should not be TANGO specific as is at the moment.

Subclasses should add all the attributes to this set that users should be able to adjust via a user interface at simulation runtime, also initialise the last_val attribute with the initial quantity value.

```python
adjustable_attributes = frozenset(['last_update_time', 'last_val'])

default_val(t)
    Set a default value of 0 to the quantity
    t [float] Time to update quantity

next_val(t)
    Return the next simulated value for simulation time at t seconds
    Must update attributes last_val with the new value and last_update_time with the simulation time
    t [float] Time to update quantity

set_val(val, t)
    Set a value to the quantity
    t [float] Time to update quantity
    val [int/float/string] Value to update quantity

tango_simlib.quantities.register_quantity_class(cls)
```

### 1.5.4 tango_simlib.sim_test_interface module

### 1.5.5 tango_simlib.tango_launcher module

### 1.5.6 tango_simlib.tango_sim_generator module

### 1.6 tango_simlib.utilities modules

#### 1.6.1 tango_simlib.utilities.base_parser module

```python
class tango_simlib.utilities.base_parser.Parser
    Bases: object

    get_device_attribute_metadata()

    get_device_cmd_override_metadata()

    get_device_command_metadata()

    get_device_properties_metadata(property_group)

    parse(data_file)
```
1.6.2 tango_simlib.utilities.fandango_json_parser module

1.6.3 tango_simlib.utilities.helper_module module

1.6.4 tango_simlib.utilities.simsdd_xml_json_parser module

1.6.5 tango_simlib.utilities.sim_xmi_parser module

1.6.6 tango_simlib.utilities.simdd_json_parser module

1.6.7 tango_simlib.utilities.testutils module

class tango_simlib.utilities.testutils.ClassCleanupUnittestMixin
    Bases: object

    Implement class-level setup/deardown semantics that emulate addCleanup()
    Subclasses can define a setUpClassWithCleanup() method that wraps addCleanup such that cls.addCleanup()
    can be used to add cleanup methods that will be called at class tear-down time.

classmethod addCleanupClass(function, *args, **kwargs)
    Add a cleanup that will be called at class tear-down time

classmethod doCleanupsClass()
    Run class-level cleansups registered with cls.addCleanupClass()

classmethod setUpClass()
    Call setUpClassWithCleanup with cls.addCleanup for class-level cleanup
    Any exceptions raised during cls.setUpClassWithCleanup will result in the cleanups registered up to that
    point being called before logging the exception with traceback.

classmethod setUpClassWithCleanup()
    Do class-level setup and ensure that cleanup functions are called
    It is intended that subclasses override this class method
    In this method calls to cls.addCleanup is forwarded to cls.addCleanupClass, which means callables regist-
    ered with cls.addCleanup() is added to the class-level cleanup function stack.

classmethod tearDownClass()

tango_simlib.utilities.testutils.cleanup_tempdir(test_instance, *mktemp_args, **mktemp_kwargs)
    Return filename of a new tempfile and add cleanup callback to test_instance.
    Will not raise an error if the directory is not present when trying to delete.
    Extra args and kwargs are passed on to the tempfile.mktemp call

        tango_simlib.utilities.testutils.cleanup_tempfile(test_instance, unlink=False, *mkstemp_args, **mkstemp_kwargs)
        Return filename of a new tempfile and add cleanup callback to test_instance.
        Will not raise an error if the file is not present when trying to delete.
        If unlink=True the actual temp file will be deleted immediately. This is useful if you want to check behaviour in
        absence of a named file.
        Extra args and kwargs are passed on to the tempfile.mkstemp call.
tango_simlib.utilities.testutils.disable_attributes_polling(test_case, device_proxy, device_server, attributes)

Disable polling for a tango device server, re-enable at end of test.

tango_simlib.utilities.testutils.set_attributes_polling(test_case, device_proxy, device_server, poll_periods)

Set attribute polling and restore after test.

**test_case** [unittest.TestCase instance] Unit test case class instance

**device_proxy** [PyTango.DeviceProxy instance] The Tango device proxy instance

**device_server** [PyTango.Device instance] The instance of the device class `device_proxy` is talking to

**poll_periods** [dict {"attribute_name":[poll_period]}] `poll_period` in milliseconds as per Tango APIs, 0 or falsy to disable polling.

**restore_polling** [function] This function can be used to restore polling if it is to happen before the end of the test. Should be idempotent if only one `set_attributes_polling()` is called per test.

1.7 tango_simlib.examples modules

1.7.1 tango_simlib.examples.mkat_vds module

1.7.2 tango_simlib.examples.weather1 module

1.7.3 tango_simlib.examples.weather2 module

1.7.4 tango_simlib.examples.weather3 module

1.7.5 tango_simlib.examples.override_class module

An example of the user-defined override class.

**exception** `tango_simlib.examples.override_class.DishSimError`

| Bases: exceptions.Exception |

Raised when a Dish simulator action could not be executed.

**class** `tango_simlib.examples.override_class.OverrideDish`

| Bases: object |

| AZIM_DRIVE_MAX_RATE = 2.0 |

| ELEV_DRIVE_MAX_RATE = 1.0 |

**action_configureband1** (model, tango_dev=None, data_input=None)

This command triggers the Dish to transition to the CONFIGURE Dish Element Mode, and returns to the caller. To configure the Dish to operate in frequency band 1. On completion of the band configuration, Dish will automatically revert to the previous Dish mode (OPERATE or STANDBY-FP).

**data_input**: str timestamp
**action_configureband2** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the CONFIGURE Dish Element Mode, and returns to the caller. To configure the Dish to operate in frequency band 2. On completion of the band configuration, Dish will automatically revert to the previous Dish mode (OPERATE or STANDBY-FP).

**data_input**: `str` timestamp

**action_configureband3** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the CONFIGURE Dish Element Mode, and returns to the caller. To configure the Dish to operate in frequency band 3. On completion of the band configuration, Dish will automatically revert to the previous Dish mode (OPERATE or STANDBY-FP).

**data_input**: `str` timestamp

**action_configureband4** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the CONFIGURE Dish Element Mode, and returns to the caller. To configure the Dish to operate in frequency band 4. On completion of the band configuration, Dish will automatically revert to the previous Dish mode (OPERATE or STANDBY-FP).

**data_input**: `str` timestamp

**action_configureband5** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the CONFIGURE Dish Element Mode, and returns to the caller. To configure the Dish to operate in frequency band 5. On completion of the band configuration, Dish will automatically revert to the previous Dish mode (OPERATE or STANDBY-FP).

**data_input**: `str` timestamp

**action_lowpower** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the LOW power state. All subsystems go into a low power state to power only the essential equipment. Specifically the Helium compressor will be set to a low power consumption, and the drives will be disabled. When issued a STOW command while in LOW power, the DS controller should be able to turn the drives on, stow the dish and turn the drives off again. The purpose of this mode is to enable the observatory to perform power management (load curtailment), and also to conserve energy for non-operating dishes.

**data_input**: None

**action_setmaintenancemodemode** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the MAINTENANCE Dish Element Mode, and returns to the caller. To go into a state that is safe to approach the Dish by a maintainer, and to enable the Engineering interface to allow direct access to low level control and monitoring by engineers and maintainers. This mode will also enable engineers and maintainers to upgrade SW and FW. Dish also enters this mode when an emergency stop button is pressed.

**data_input**: None

**action_setoperatemode** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the OPERATE Dish Element Mode, and returns to the caller. This mode fulfils the main purpose of the Dish, which is to point to designated directions while capturing data and transmitting it to CSP.

**data_input**: None

**action_setstandbyfpmode** *(model, tango_dev=None, data_input=None)*

This command triggers the Dish to transition to the STANDBY-FP Dish Element Mode, and returns to the caller. To prepare all subsystems for active observation, once a command is received by TM to go to the FULL_POWER mode.

**data_input**: None
**action_setstandbylpmode** (*model, tango_dev=None, data_input=None*)

This command triggers the Dish to transition to the STANDBY-LP Dish Element Mode, and returns to the caller. Standby_LP is the default mode when the Dish is configured for low power consumption, and is the mode wherein Dish ends after a start up procedure.

```python
data_input: None
```

**action_setstowmode** (*model, tango_dev=None, data_input=None*)

This command triggers the Dish to transition to the STOW Dish Element Mode, and returns to the caller. To point the dish in a direction that minimises the wind loads on the structure, for survival in strong wind conditions. The Dish is able to observe in the stow position, for the purpose of transient detection.

```python
data_input: None
```

**action_slew** (*model, tango_dev=None, data_input=None*)

The Dish is tracking the commanded pointing positions within the specified TRACK pointing accuracy.

```python
data_input: list [Timestamp] [azimuth] [elevation]
```

**pre_update** (*sim_model, sim_time, dt*)

**class** `tango_simlib.examples.override_class.OverrideVds`

Bases: `object`

An example of the override class for the TANGO device class ‘MKATVDS’. It provides all the implementations of the command handler functions for the commands specified in the POGO generated XMI data description file.

**action_camerapoweron** (*model, tango_dev=None, data_input=None*)

Switch camera electronics on or off.

```python
data_input[0] [str] ‘on’ or ‘off’ value.
```

**action_floodlighton** (*model, tango_dev=None, data_input=None*)

Set floodlight to a on or off.

```python
data_input[0] [str] ‘on’ or ‘off’ value.
```

**action_focus** (*model, tango_dev=None, data_input=None*)

Focuses camera to a specified direction or specified position.

```python
data_input[0] [str] focus direction e.g. ‘far’, ‘near’, ‘to’.
data_input[1] [str] Optional argument, focus position; a stringified integer numeral.
```

**action_pan** (*model, tango_dev=None, data_input=None*)

Drive camera to a pan direction(left or right) or pan to specified position.

```python
data_input[0] [str] pan direction e.g. ‘left’, ‘right’, ‘to’.
data_input[1] [str] Optional argument, pan position; a stringified integer numeral.
```

**action_presetclear** (*model, tango_dev=None, data_input=None*)

Clear the specified preset.

```python
data_input[0] [str] receptor name (from m000 to m063).
```

**action_presetgoto** (*model, tango_dev=None, data_input=None*)

Go to preset stored position(pan, tilt, zoom).

```python
data_input[0] [str] receptor name (from m000 to m063).
```

**action_presetset** (*model, tango_dev=None, data_input=None*)

Set the position which the camera is at currently as preset position.

```python
data_input[0] [str] receptor name (from m000 to m063).
```
**action_stop** (model, tango_dev=None, data_input=None)
Stop camera.

**action_tilt** (model, tango_dev=None, data_input=None)
Drive camera to a tilt direction or specified position.

*data_input[0]* [str] tilt_direction e.g. 'up', 'down', 'to'.

*data_input[1]* [str] Optional argument, tilt_position; a stringified integer numeral.

**action_trapupdate** (model, tango_dev=None, data_input=None)
Update trap. this request is called by a script.

*data_input* [str] Trap update from a script (8, 'on').

**action_zoom** (model, tango_dev=None, data_input=None)
Zoom camera to a specified direction or specified position.

*data_input[0]* [str] zoom_direction e.g. 'tele', 'wide', 'to'.

*data_input[1]* [str] Optional argument, zoom_position; a stringified numeral.

**class** tango_simlib.examples.override_class.OverrideWeather

*Bases:* object

An example of the override class for the TANGO device class 'Weather'. It provides all the implementations of the command handler functions for the commands specified in the POGO generated XMI data description file.

**action_add** (model, tango_dev=None, data_input=None)
Add two or more numbers together and return their sum.

**action_multiplystringby3** (model, tango_dev=None, data_input=None)
 Takes a string and multiplies it by a constant integer value of 3.

**action_off** (model, tango_dev=None, data_input=None)
Changes the State of the device to OFF.

**action_on** (model, tango_dev=None, data_input=None)
Changes the State of the device to ON.

**class** tango_simlib.examples.override_class.OverrideWeatherSimControl

*Bases:* object

An example of the override class for the TANGO device class 'SimControl'. It provides all the implementations of the command handler functions for the commands required to stimulate a running TANGO device in real time.

**test_action_setattributemaxvalue** (model, tango_dev=None, data_input=None)
This command sets an attribute value to its maximum value to set its quality to Alarm state to warning.

**test_action_setoffrainstorm** (model, tango_dev=None, data_input=None)

**test_action_setoffwindstorm** (model, tango_dev=None, data_input=None)

**test_action_simulatefaultdevicestate** (model, tango_dev=None, data_input=None)
This command sets the current device state to fault/on.

**test_action_stimulateattributeconfigurationerror** (model, tango_dev=None, data_input=None)
This command sets the attribute maximum allowed value to be the same as that minimum allowed value.

**test_action_stopquantitiesimulation** (model, tango_dev=None, data_input=None)
Totally sets the simulated quantities’ values to a constant value of zero.

**test_action_stoptrainfall** (model, tango_dev=None, data_input=None)
Totally sets the simulated quantity rainfall to a constant value of zero.
test_action_stoprainstorm(model, tango_dev=None, data_input=None)

test_action_stopwindstorm(model, tango_dev=None, data_input=None)

exception tango_simlib.examples.override_class.VdsSimError
    Bases: exceptions.Exception
    Raised when a Video Display System simulator action could not be executed.

exception tango_simlib.examples.override_class.WeatherSimError
    Bases: exceptions.Exception
    Raised when a Weather simulator action could not be executed.

1.8 License

This project is licensed under the BSD 3-Clause License - see https://opensource.org/licenses/BSD-3-Clause for details.

1.9 Contribute

Contributions are always welcome! Please ensure that you adhere to our coding standards CAM_Style_guide.

1.10 Contact us

SKA South Africa

cam@ska.ac.za

https://www.ska.ac.za/
CHAPTER 2

Indices and tables

- genindex
- modindex
- search
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tango_simlib.examples.override_class</td>
<td>10</td>
</tr>
<tr>
<td>tango_simlib.quantities</td>
<td>7</td>
</tr>
<tr>
<td>tango_simlib.utilities.base_parser</td>
<td>8</td>
</tr>
<tr>
<td>tango_simlib.utilities.testutils</td>
<td>9</td>
</tr>
</tbody>
</table>
Index

A

action_add() (tango_simlib.examples.override_class.OverrideWeather method), 13
action_cameraloweron() (tango_simlib.examples.override_class.OverrideVds method), 12
action_configureband1() (tango_simlib.examples.override_class.OverrideDish method), 10
action_configureband2() (tango_simlib.examples.override_class.OverrideDish method), 10
action_configureband3() (tango_simlib.examples.override_class.OverrideDish method), 11
action_configureband4() (tango_simlib.examples.override_class.OverrideDish method), 11
action_configureband5() (tango_simlib.examples.override_class.OverrideDish method), 11
action_floodlighton() (tango_simlib.examples.override_class.OverrideVds method), 12
action_focus() (tango_simlib.examples.override_class.OverrideVds method), 12
action_lowpower() (tango_simlib.examples.override_class.OverrideDish method), 11
action_multiplystringby3() (tango_simlib.examples.override_class.OverrideWeather method), 13
action_off() (tango_simlib.examples.override_class.OverrideWeather method), 13
action_on() (tango_simlib.examples.override_class.OverrideWeather method), 13
action_pan() (tango_simlib.examples.override_class.OverrideVds method), 12
action_presetclear() (tango_simlib.examples.override_class.OverrideVds method), 12
action_presetgoto() (tango_simlib.examples.override_class.OverrideVds method), 12
action_presetset() (tango_simlib.examples.override_class.OverrideVds method), 12
action_setmaintenancemode() (tango_simlib.examples.override_class.OverrideDish method), 11
action_setoperatemode() (tango_simlib.examples.override_class.OverrideDish method), 11
action_setstandbyfpmode() (tango_simlib.examples.override_class.OverrideDish method), 11
action_setstandbylpmode() (tango_simlib.examples.override_class.OverrideDish method), 11
action_setstowmode() (tango_simlib.examples.override_class.OverrideDish method), 11
action_slew() (tango_simlib.examples.override_class.OverrideDish method), 12
action_stop() (tango_simlib.examples.override_class.OverrideVds method), 12
action_tilt() (tango_simlib.examples.override_class.OverrideVds method), 13
action_trapupdate() (tango_simlib.examples.override_class.OverrideVds method), 13
action_zoom() (tango_simlib.examples.override_class.OverrideVds method), 13
addCleanupClass() (tango_simlib.utilities.testutils.ClassCleanupUnittestMixin class method), 9
adjustable_attributes (tango_simlib.quantities.GaussianSlewLimited attribute), 7
adjustable_attributes (tango_simlib.quantities.Quantity attribute), 8
AZIM_DRIVE_MAX_RATE (tango_simlib.examples.override_class.OverrideDish attribute), 10
default_val() (tango_simlib.quantities.ConstantQuantity method), 7
default_val() (tango_simlib.quantities.Quantity method), 8
disable_attributes_polling() (in module tango_simlib.utilities.testutils), 9
DishSimError, 10
doCleanupsClass() (tango_simlib.utilities.testutils.ClassCleanupUnittestMixin class method), 9

elev_drive_max_rate
tango_simlib.examples.override_class.OverrideDish attribute), 10
G
GaussianSlewLimited (class in tango_simlib.quantities), 7
get_device_attribute_metadata() (tango_simlib.utilities.base_parser.Parser method), 8
get_device_cmd_override_metadata() (tango_simlib.utilities.base_parser.Parser method), 8
get_device_command_metadata() (tango_simlib.utilities.base_parser.Parser method), 8
get_device_properties_metadata() (tango_simlib.utilities.base_parser.Parser method), 8

N
next_val() (tango_simlib.quantities.ConstantQuantity method), 7
next_val() (tango_simlib.quantities.GaussianSlewLimited method), 7
next_val() (tango_simlib.quantities.Quantity method), 8

O
overrideDish (class in tango_simlib.examples.override_class), 10
overrideVds (class in tango_simlib.examples.override_class), 12
overrideWeather (class in tango_simlib.examples.override_class), 13
overrideWeatherSimControl (class in tango_simlib.examples.override_class), 13

P
parse() (tango_simlib.utilities.base_parser.Parser method), 8
Parser (class in tango_simlib.utilities.base_parser), 8
pre_update() (tango_simlib.examples.override_class.OverrideDish method), 12
Q
Quantity (class in tango_simlib.quantities), 7
R
register_quantity_class() (in module tango_simlib.quantities), 8
S
set_attributes_polling() (in module tango_simlib.utilities.testutils), 10
set_val() (tango_simlib.quantities.Quantity method), 8
setUpClass() (tango_simlib.utilities.testutils.ClassCleanupUnittestMixin class method), 9
setUpClassWithCleanup() (tango_simlib.utilities.testutils.ClassCleanupUnittestMixin class method), 9

tango_simlib.examples.override_class (module), 10
tango_simlib.quantities (module), 7
tango_simlib.utilities.base_parser (module), 8
tango_simlib.utilities.testutils (module), 9
tearDownClass() (tango_simlib.utilities.testutils.ClassCleanupUnittestMixin method), 9
test_action_setattributemaxvalue()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_setoffrainstorm()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_setoffwindstorm()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_simulatefaultdevicestate()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_stimulateattributeconfigurationerror()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_stopquantitysimulation()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_stoprainfall()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 13
test_action_stopwindstorm()
(tango_simlib.examples.override_class.OverrideWeatherSimControl method), 14

P
Parser (class in tango_simlib.utilities.base_parser), 8
V
VdsSimError, 14

W
WeatherSimError, 14