Main Telescope Slew simulation: Setup notebook

....

This notebook does slew simulations, and check all aos components (M1M3, M2, hexapods) behavior during the slew-and-track process

This is expected to work both for SUMMIT and NCSA

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import rubin_jupyter_utils.lab.notebook as nb
nb.utils.get_node()
```

```
/tmp/ipykernel_31037/1665379685.py:2: DeprecationWarning: Call to deprecated f
unction (or staticmethod) get_node. (Please use lsst.rsp.get_node())
  nb.utils.get_node()
```

Out[2]:
```
'yagan03'
```

In [3]:
```
import os
import sys
import asyncio
import logging

import pandas as pd

from matplotlib import pyplot as plt

from lsst.ts import salobj
from lsst.ts.observatory.control.maintel.mtcs import MTCS
```

```
lsst.ts.utils.tai INFO: Update leap second table
lsst.ts.utils.tai INFO: current_tai uses the system TAI clock
```

In [4]:
```
summit = 1 #use this for summit testing
# summit = 0 #use this for NCSA
```

Check environment setup

The following cell will print some of the basic DDS configutions.

In [5]:
```
print(os.environ["OSPL_URI"])
print(os.environ["LSST_DDS_PARTITION_PREFIX"])
print(os.environ.get("LSST_DDS_DOMAIN_ID", "Expected, not set."))
```

```
file:///home/b1quint/WORK/ts_ddsconfig/config/ospl-shmem.xml
summit
0
```

Setup logging

Setup logging in debug mode and create a logger to use on the notebook.

In [6]:
```
logging.basicConfig(format="%(name)s:%(message)s", level=logging.DEBUG)
```

In [7]:
```
log = logging.getLogger("setup")
log.level = logging.DEBUG
```

Starting communication resources

We start by creating a domain and later instantiate the MTCS class. We will use the class to startup the components.

In [8]:
```
domain = salobj.Domain()
```

In [9]:
```
mtcs = MTCS(domain=domain, log=log)
mtcs.set_rem_loglevel(40)
```

```
setup.MTCS DEBUG: mtmount: Adding all resources.
setup.MTCS DEBUG: mtptg: Adding all resources.
setup.MTCS DEBUG: mtaos: Adding all resources.
setup.MTCS DEBUG: mtm1m3: Adding all resources.
setup.MTCS DEBUG: mtm2: Adding all resources.
setup.MTCS DEBUG: mthexapod_1: Adding all resources.
setup.MTCS DEBUG: mthexapod_2: Adding all resources.
setup.MTCS DEBUG: mtrotator: Adding all resources.
setup.MTCS DEBUG: mtdome: Adding all resources.
setup.MTCS DEBUG: mtdometrajectory: Adding all resources.
```

In [10]:
```
await mtcs.start_task
```

```
MTHexapod INFO: Read historical data in 0.06 sec
MTHexapod INFO: Read historical data in 0.12 sec
```

Out[10]:
```
[None, None, None, None, None, None, None, None, None, None]
```

```
MTMount.elevation ERROR: tel_elevation DDS read queue is full (100 element
s); data may be lost
MTMount.azimuth ERROR: tel_azimuth DDS read queue is full (100 elements); d
ata may be lost
```

Starting components

From now on we will start the various components of the MTAOS. You may wonder why are we not simply sending all CSCs to ENABLED state in one go, as we usually do on other systems.

The answer is that the MTCS components have some initilization dependencies that need to be observed for the components to be enabled properly. We will describe these as we work our way the initialization steps.

Starting MTPtg

We start by making sure the pointing component is alive, by waiting for a heartbeat. Next we enable the component using `mtcs.set_state` method.

We select to start with the `MTPtg` mainly because, of all components of the `MTCS` it is the only pure-software components. As such the `MTPtg` is pretty independent and can be brought to enabled in any condition.

It is also worth noticed that, as a pure-software component, the `MTPtg` does not have a simulation mode.

Furthermore, as you will notice below, we are not checking the software version of the `MTPtg`, mainly because the component is currently not sending this information.

In [11]:
```
await mtcs.next_heartbeat("mtptg")
```

Out[11]:
```
<ddsutil.MTPtg_logevent_heartbeat_b28358a6 at 0x7fc33851edf0>
```

In [12]:
```
await mtcs.set_state(salobj.State.ENABLED, components=["mtptg"])
```

```
setup.MTCS DEBUG: [mtptg]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <State.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

Starting MTMount

This is one case where the initialization order is important.

The MTMount needs to be enabled before we enable the MTRotator. The reason is that the MTRotator needs to know the position of the Camera Cable Wrap (CCW), which is provided by the MTMount, before it can be enable. If the MTRotator does not receive the position of the CCW, it will immediatelly activate the breaks and transition to FAULT state.

We start by verifying that the CSC is sending heartbeats.

In [13]:
```
await mtcs.next_heartbeat("mtmount")
```

Out[13]:
```
<ddsutil.MTMount_logevent_heartbeat_d373cb25 at 0x7fc33888de20>
```

Now we can enable the CSC.

In [14]:
```
await mtcs.set_state(salobj.State.ENABLED, components=["mtmount"])
```

```
setup.MTCS DEBUG: [mtmount]::[<State.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

Perform some basic checks

The following are a couple of sanity checks we routinely perform when starting the MTMount.

We check if the CSC is running in simulation mode and then the version of the CSC.

Finally, we verify that the camera cable wrap following is enabled.

In [15]:
```
mtmount_simulation_mode = await mtcs.get_simulation_mode(["mtmount"])

mode = mtmount_simulation_mode["mtmount"].mode
timestamp = pd.to_datetime(mtmount_simulation_mode["mtmount"].private_sndStamp,

log.debug(
    f"MTMount simulation mode: {mode} @ {timestamp}"
)
```

```
setup DEBUG: MTMount simulation mode: 0 @ 2022-05-20 14:43:00.771810560
```

In [16]:
```
mtmount_software_versions = await mtcs.get_software_versions(["mtmount"])

csc_version = mtmount_software_versions["mtmount"].cscVersion
timestamp = pd.to_datetime(mtmount_software_versions["mtmount"].private_sndStan

log.debug(
    f"MTMount software version: {csc_version} @ {timestamp}",
)
```

```
setup DEBUG: MTMount software version: 0.4.0rc2.dev194+g7e3b2bc.d20220513 @
2022-05-20 14:43:00.772554496
```

In [17]:
```
mtmount_ccw_following = await mtcs.rem.mtmount.evt_cameraCableWrapFollowing.age

timestamp = pd.to_datetime(mtmount_ccw_following.private_sndStamp, unit='s')

if mtmount_ccw_following.enabled:
    log.debug(f"CCW following mode enabled: {mtmount_ccw_following.enabled} @ {
else:
    await mtcs.set_state(salobj.State.DISABLED, ["mtmount"])
    raise RuntimeError(
        "CCW following mode not enabled. Usually this means that the MTMount cc
        "not see telemetry from the rotator when it was enabled. To correct thi
        "make sure the MTRotator telemetry is being published, then execute the
        "MTMount CSC will be left in DISABLED state."
        )
```

```
setup DEBUG: CCW following mode enabled: 1 @ 2022-06-02 16:59:54.982228736.
```

In [18]:
```
# We want to turn off CCW following if MTMount is in Simulation Mode
if mtmount_simulation_mode["mtmount"].mode:
    await mtcs.disable_ccw_following()
```

Starting Rotator

In [19]:
```
await mtcs.next_heartbeat("mtrotator")
```

Out[19]:
```
<ddsutil.MTRotator_logevent_heartbeat_6ca7fbf4 at 0x7fc339117160>
```

In [20]:
```
await mtcs.set_state(salobj.State.ENABLED, components=["mtrotator"])
```

```
setup.MTCS DEBUG: [mtrotator]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <S
tate.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

Perform some basic checks

The following is a few sanity checks we routinely perform to verify the system integrity at this stage.

In [21]:
```
mtrotator_simulation_mode = await mtcs.get_simulation_mode(["mtrotator"])

mode = mtrotator_simulation_mode["mtrotator"].mode
timestamp = pd.to_datetime(mtrotator_simulation_mode["mtrotator"].private_sndSt

log.debug(
    f"MTRotator simulation mode: {mode} @ {timestamp}"
)
```

```
setup DEBUG: MTRotator simulation mode: 0 @ 2022-05-27 15:17:37.316813824
```

In [22]:
```
mtrotator_software_versions = await mtcs.get_software_versions(["mtrotator"])

csc_version = mtrotator_software_versions["mtrotator"].cscVersion
timestamp = pd.to_datetime(mtrotator_software_versions["mtrotator"].private_snc

log.debug(
    f"MTRotator software version: {csc_version} @ {timestamp}",
)
```

```
setup DEBUG: MTRotator software version: 0.25.0a1 @ 2022-05-27 15:17:37.317
080832
```

In [23]:
```
elevation = await mtcs.rem.mtmount.tel_elevation.next(flush=True, timeout=5)
azimuth = await mtcs.rem.mtmount.tel_azimuth.next(flush=True, timeout=5)
ccw = await mtcs.rem.mtmount.tel_cameraCableWrap.next(flush=True, timeout=5)
rotator = await mtcs.rem.mtrotator.tel_rotation.next(flush=True, timeout=5)

log.info(f"mount elevation Angle = {elevation.actualPosition}")
log.info(f"mount azimuth angle = {azimuth.actualPosition}")
log.info(f"CCW angle = {ccw.actualPosition}. Needs to be within 2.2 deg of rota
log.info(f"rot angle = {rotator.actualPosition} diff = {rotator.actualPosition
```

```
setup INFO: mount elevation Angle = 60.8077748965249
setup INFO: mount azimuth angle = 120.6394843716448
setup INFO: CCW angle = 0.0855. Needs to be within 2.2 deg of rotator angle
setup INFO: rot angle = 0.09860114575079137 diff = 0.013101145750791368
```

CCW telemetry too old

This warning message may appear in the `MTRotator` in a couple different conditions.

The most common occurence is when the `MTMount` component is not publishing the CCW telemetry. This should be rectified by enabling the CSC, as we've done on the section above, and is one of the reasons we enable `MTMount` before the `MTRotator`.

The less common but more critical condition is when the clock on the `MTMount` controller is out of sync with the observatory clock server. In this case, the `timestamp` attribute, used by the `MTRotator` to determine the relevant time for the published telemetry, will be out of sync and we won't be able to operate the system.

You can use the cell below to determine whether this is the case or not. If so, you need to contact IT or someone with knowledge about the `MTMount` low level controller to fix the time synchronization issue.

In [24]:
```
ccw = await mtcs.rem.mtmount.tel_cameraCableWrap.next(flush=True, timeout=5)
rotator = await mtcs.rem.mtrotator.tel_rotation.next(flush=True, timeout=5)

ccw_snd_stamp = pd.to_datetime(ccw.private_sndStamp, unit='s')
ccw_timestamp = pd.to_datetime(ccw.timestamp, unit='s')
ccw_actual_position = ccw.actualPosition

rotator_snd_stamp = pd.to_datetime(rotator.private_sndStamp, unit='s')
rotator_timestamp = pd.to_datetime(rotator.timestamp, unit='s')
rotator_actual_position = rotator.actualPosition

log.info(
    f"CCW:: snd_stamp={ccw_snd_stamp} timestamp={ccw_timestamp} actual positior
    )
log.info(
    f"Rotator:: snd_stamp={rotator_snd_stamp} timestamp={rotator_timestamp} act
    )

ccw_telemetry_maximum_age = pd.to_timedelta(1.0, unit='s')

if abs(ccw_snd_stamp - ccw_timestamp) > ccw_telemetry_maximum_age:
    log.warning(
        f"CCW timestamp out of sync by {abs(ccw_snd_stamp - ccw_timestamp)}s. '
        "System may not work. Check clock synchronization in MTMount low level
        )
```

```
setup INFO: CCW:: snd_stamp=2022-06-03 14:49:40.115448320 timestamp=2022-06
-03 14:49:40.011869952 actual position=0.0855
setup INFO: Rotator:: snd_stamp=2022-06-03 14:49:40.141991680 timestamp=202
2-06-03 14:49:40.041159168 actual position=0.09860133740446031
```

Clearing error in MTRotator

If the MTRotator is in FAULT state, you need to send the `clearError` command before transitioning it back to `ENABLED`.

This is a particularity of the `MTRotator` (and `MTHexapod`) that violates our state machine.

In [ ]:
```
if False:
    await mtcs.rem.mtrotator.cmd_clearError.set_start()
```

Checkpoint

At this point the system is ready for exercicing slew activities, without involving the optical components.

Starting M1M3 (Mount telemetry mode)

If running the test on level 3 and if M1M3 is configured to listen for the mount telemetry, we firt need to make sure the `MTMount` is pointing to zenith.

The reason is that `M1M3` is in a fixed position and, when we try to enabled/raise it, the will check the inclinometer data against the elevation data. If they differ by more than a couple degrees the process will fail.

Once M1M3 is mounted on the telescope and we are operating the actual mount, instead of in simulation mode, this will not be necessary.

In [25]:
```
await mtcs.rem.mtmount.cmd_moveToTarget.set_start(azimuth=0, elevation=90)
```

Out[25]:
```
<ddsutil.MTMount_ackcmd_d68fb318 at 0x7fc2d6701ca0>
```

In [26]:
```
await mtcs.next_heartbeat("mtm1m3")
```

Out[26]:
```
<ddsutil.MTM1M3_logevent_heartbeat_d6c09f79 at 0x7fc3388db5e0>
```

In [29]:
```
await mtcs.set_state(
    state=salobj.State.ENABLED,
    components=["mtm1m3"],
    overrides = {"mtm1m3": 'Default'}
    )
```

```
setup.MTCS DEBUG: [mtm1m3]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <State.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

Raise m1m3

Now that m1m3 is enabled we can raise it.

The following has a trick to allow raising the m1m3 in the background and give control back to the notebook. If, in middle of the process, you need to abort the operation you can still do it from the notebooks.

Once you execute the cell bellow you will notice that the log messages will appear below the cell, but you can also see that the cell will be masked as "finished executing". That means, instead of seeing an `*` you will see the number of the cell. This is because the operation is running in the background and we have control over the notebook to execute additional cells.

In [30]:
```
task_raise_m1m3 = asyncio.create_task(mtcs.raise_m1m3())
```

```
setup.MTCS DEBUG: M1M3 current detailed state {<DetailedState.PARKEDENGINEERING: 9>, <DetailedState.PARKED: 5>}, executing command...
setup.MTCS DEBUG: process as completed...
setup.MTCS DEBUG: M1M3 detailed state 6
setup.MTCS DEBUG: mtm1m3: <State.ENABLED: 2>
```

The next cell contain a command to abort the raise operation initiated in the background on the cell above. Note that the command to execute the abort operation is encapsulated by an `if False`. This is to prevent the command from executing if the notebook is being executed by papermill or by accident.

If you need to abort the operation change the if statement to `if True`.

In [ ]:
```
if False:
    await mtcs.abort_raise_m1m3()
```

The next cell will wait for the raise_m1m3 command to finish executing. This is to make sure a batch processing of the notebook won't proceed until the raise operation is completed.

In [31]:
```
await task_raise_m1m3
```

```
setup.MTCS DEBUG: mtm1m3: <State.ENABLED: 2>
setup.MTCS DEBUG: M1M3 detailed state 7
```

In [32]:
```
await mtcs.enable_m1m3_balance_system()
```

```
setup.MTCS DEBUG: Enabling hardpoint corrections.
```

In [33]:
```
await mtcs.reset_m1m3_forces()
```

Starting M2

In [34]:
```
await mtcs.next_heartbeat("mtm2")
```

Out[34]:
```
<ddsutil.MTM2_logevent_heartbeat_c8b944e6 at 0x7fc2cd4897c0>
```

In [42]:
```
await mtcs.set_state(
    state=salobj.State.ENABLED,
    components=["mtm2"]
    )
```

```
setup.MTCS DEBUG: [mtm2]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <State.ENABLED: 2>]
```

```
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

Prepare M2 for operation

Switch on m2 force balance system and reset m2 forces.

In [43]:
```
await mtcs.enable_m2_balance_system()
```

```
setup.MTCS INFO: M2 force balance system already enabled. Nothing to do.
```

In [44]:
```
await mtcs.reset_m2_forces()
```

Starting Camera Hexapod

In [45]:
```
await mtcs.next_heartbeat("mthexapod_1")
```

Out[45]:
```
<ddsutil.MTHexapod_logevent_heartbeat_ae564757 at 0x7fc2d66c0d60>
```

The documentation below is now deprecated. We are leaving it just for the sake of keeping the information alive. You can simply run the commands that follow.

> The command bellow to enable the Camera Hexapod should work, in general. Nevertheless, we found an issue with the interaction between the low level controller and the CSC that was causing it to fail from time to time.
> The error report can be found in DM-31111.
> Until this ticket is worked on you may encounter failures when executing the cell below. You can continue by running the cell again.
>
> In addition to the ticket above, the software of camera hexapod controller and EUI v1.2.0 on summit require the `mthexapod_1` to be in `DISABLED` state when setting the command source to DDS/CSC.

In [46]:
```
await mtcs.set_state(
    state=salobj.State.ENABLED,
    components=["mthexapod_1"]
    )
```

```
setup.MTCS DEBUG: [mthexapod_1]::[<State.STANDBY: 5>, <State.DISABLED: 1>,
 <State.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

In [47]:
```
mthexapod_1_simulation_mode = await mtcs.get_simulation_mode(["mthexapod_1"])

mode = mthexapod_1_simulation_mode["mthexapod_1"].mode
timestamp = pd.to_datetime(mthexapod_1_simulation_mode["mthexapod_1"].private_s

log.debug(
    f"Camera Hexapod simulation mode: {mode} @ {timestamp}"
)
```

```
setup DEBUG: Camera Hexapod simulation mode: 0 @ 2022-05-12 19:24:16.225221
376
```

In [48]:
```
mthexapod_1_software_versions = await mtcs.get_software_versions(["mthexapod_1'

csc_version = mthexapod_1_software_versions["mthexapod_1"].cscVersion
timestamp = pd.to_datetime(mthexapod_1_software_versions["mthexapod_1"].private

log.debug(
    f"Camera Hexapod software version: {csc_version} @ {timestamp}",
)
```

```
setup DEBUG: Camera Hexapod software version: 0.26.0 @ 2022-05-12 19:24:16.
225595904
```

In [ ]:
```
if False:
    await mtcs.rem.mthexapod_1.cmd_clearError.set_start()
```

In [49]:
```
await mtcs.enable_compensation_mode(component="mthexapod_1")
```

```
setup.MTCS DEBUG: Setting mthexapod_1 compensation mode from False to True.
```

In [50]:
```
await mtcs.reset_camera_hexapod_position()
```

```
setup.MTCS INFO: Camera Hexapod compensation mode enabled. Move will offset
with respect to LUT.
setup.MTCS DEBUG: Wait for Camera Hexapod in position event.
setup.MTCS DEBUG: Camera Hexapod in position: True.
setup.MTCS DEBUG: Camera Hexapod already in position. Handling potential ra
ce condition.
setup.MTCS INFO: Camera Hexapod in position: False.
setup.MTCS INFO: Camera Hexapod in position: True.
setup.MTCS DEBUG: Camera Hexapod in position True. Waiting settle time 5.0s
```

Starting M2 Hexapod

In [51]:
```
await mtcs.next_heartbeat("mthexapod_2")
```

Out[51]:
```
<ddsutil.MTHexapod_logevent_heartbeat_ae564757 at 0x7fc2d67d9ca0>
```

We have been mostly running the M2 Hexapod in simulation mode, because the actual hardware is mounted on the telescope. This means the M2 Hexapod is not affected by the issue we reported above for the Camera Hexapod.

In [52]:
```
await mtcs.set_state(
    state=salobj.State.ENABLED,
    components=["mthexapod_2"]
    )
```

```
setup.MTCS DEBUG: [mthexapod_2]::[<State.STANDBY: 5>, <State.DISABLED: 1>,
 <State.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

In [53]:
```
mthexapod_2_simulation_mode = await mtcs.get_simulation_mode(["mthexapod_2"])
```

```
mode = mthexapod_2_simulation_mode["mthexapod_2"].mode
timestamp = pd.to_datetime(mthexapod_2_simulation_mode["mthexapod_2"].private_s

log.debug(
    f"M2 Hexapod simulation mode: {mode} @ {timestamp}"
)
```

```
setup DEBUG: M2 Hexapod simulation mode: 1 @ 2022-05-13 16:28:18.896282624
```

In [54]:
```
mthexapod_2_software_versions = await mtcs.get_software_versions(["mthexapod_2'

csc_version = mthexapod_2_software_versions["mthexapod_2"].cscVersion
timestamp = pd.to_datetime(mthexapod_2_software_versions["mthexapod_2"].private

log.debug(
    f"M2 Hexapod software version: {csc_version} @ {timestamp}",
)
```

```
setup DEBUG: M2 Hexapod software version: 0.26.0 @ 2022-05-13 16:28:18.8966
38208
```

In [55]:
```
await mtcs.enable_compensation_mode(component="mthexapod_2")
```

```
setup.MTCS DEBUG: Setting mthexapod_2 compensation mode from False to True.
```

In [56]:
```
await mtcs.reset_camera_hexapod_position()
```

```
setup.MTCS INFO: Camera Hexapod compensation mode enabled. Move will offset
with respect to LUT.
setup.MTCS DEBUG: Wait for Camera Hexapod in position event.
setup.MTCS DEBUG: Camera Hexapod in position: True.
setup.MTCS DEBUG: Camera Hexapod already in position. Handling potential ra
ce condition.
setup.MTCS INFO: Camera Hexapod in position: False.
setup.MTCS INFO: Camera Hexapod in position: True.
setup.MTCS DEBUG: Camera Hexapod in position True. Waiting settle time 5.0s
```

Starting all other components

In [57]:
```
await mtcs.enable()
```

```
setup.MTCS INFO: Enabling all components
setup.MTCS DEBUG: Expand overrides None
setup.MTCS DEBUG: Complete overrides: {'mtmount': '', 'mtptg': '', 'mtaos':
'', 'mtm1m3': '', 'mtm2': '', 'mthexapod_1': '', 'mthexapod_2': '', 'mtrota
tor': '', 'mtdome': '', 'mtdometrajectory': ''}
setup.MTCS DEBUG: [mtmount]::[<State.ENABLED: 2>]
setup.MTCS DEBUG: [mtptg]::[<State.ENABLED: 2>]
setup.MTCS DEBUG: [mtaos]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <Stat
e.ENABLED: 2>]
setup.MTCS DEBUG: [mtm1m3]::[<State.ENABLED: 2>]
setup.MTCS DEBUG: [mtm2]::[<State.ENABLED: 2>]
setup.MTCS DEBUG: [mthexapod_1]::[<State.ENABLED: 2>]
setup.MTCS DEBUG: [mthexapod_2]::[<State.ENABLED: 2>]
setup.MTCS DEBUG: [mtrotator]::[<State.ENABLED: 2>]
```

```
setup.MTCS DEBUG: [mtdome]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <State.ENABLED: 2>]
setup.MTCS DEBUG: [mtdometrajectory]::[<State.STANDBY: 5>, <State.DISABLED: 1>, <State.ENABLED: 2>]
setup.MTCS INFO: All components in <State.ENABLED: 2>.
```

Closing MTCS and Domain

You can use the commands below to easily shut-down (send to STANDBY) all the components.

In [ ]:
```
await mtcs.set_state(salobj.State.STANDBY, components=["mtaos"])
```

In [ ]:
```
# Move this to a shutdown notebook...
await mtcs.lower_m1m3()
```

In [ ]:
```
await mtcs.set_state(salobj.State.STANDBY, components=["mtm1m3"])
```

In [ ]:
```
await mtcs.set_state(salobj.State.STANDBY, components=["mtm2"])
```

In [ ]:
```
await mtcs.set_state(salobj.State.STANDBY, components=["mthexapod_1"])
```

In [ ]:
```
await mtcs.set_state(salobj.State.STANDBY, components=["mtdometrajectory"])
```

In [ ]:
```
await mtcs.standby()
```

In [ ]:
```
await mtcs.close()
```

In [ ]:
```
await domain.close()
```