mTree Cheat Sheet

ACTOR CLASS and MES SETUP

```
from mTree.microeconomic_system.environment import Environment
from mTree.microeconomic_system.institution import Institution
from mTree.microeconomic_system.agent import Agent
from mTree.microeconomic_system.directive_decorators import *
from mTree.microeconomic_system.message import Message
    @directive_enabled_class    # And one of the following
    class yourEnvironment(Environment):
    class yourInstitution(Institution):
    class BargAgent(Agent):
```

ACTOR INFO

```
    actor_class_loc = "actor_file_name.ActorClass"
    short_name = "actor_class_loc k"   # k indicates k'th instance of actor
            note:  self.short_name   # Holds short-name of actor
    actor_type = 'environment'|'institution'|'agent'
```

MESSAGES (sent between actors)

```
    new_message = Message()
```

SEND MESSAGES

```
    .set_sender(self.myAddress)
    .set_directive(directive)
    .set_payload(payload)
    self.send(receiver_address, new_message)
    self.send_message(directive, receiver_address/shortname, payload)
    self.reminder(seconds_to_reminder = seconds_to_reminder,
            message = reminder_msg)
```

RECEIVE MESSAGES

```
    @directive_decorator("directive")
    def directive(self, message: Message):
    'start_enviornment' directive sent from mTree to environment
    .get_sender()
    .get_payload()
```

LOG

```
    self.log_message(string)   # unix_time + string -- in experiment.log
            hint – use header on string to make for easy search
    self.log_data(string)   # unix_time + "string"  -- in experiment.data
```

AUTO LOGGED MESSAGES -- in experiment.log

```
    Actor_Type (short_name) : About to enter directive: directive
    Actor_Type (short_name) : Exited directive: directive
    Message directive from short_name to short_name
```

ENVIRONMENT

```
self.get_property(key)   # JSON key in config file
self.shutdown_mes()     # shutdown mes system
```

CONFIG (your_config.json – in config folder)

```
{
"mtree_type": "mes_simulation_description",
"name":"Basic Simulation Run",
"id": "1",
"environment": actor_class_loc,
"institutions": [{"institution_name": actor_class_loc, “number”:1}, …]   #  “number” of instances
"number_of_runs": 1,
"data_logging": "json",
"agents": [{"agent_name": actor_class_loc, "number": 5}, …],
"properties": {"this_a_property":"this_is_a_property", …}
}
```

Any actor can read a config file before your actor starts running by adding a prepare method.

```
def prepare(self):
    self.value = int(self.get_property("value"))
    self.log_message(f'***<A>*** {self.role}: Initialized')
```

Use prepare() and not __init__ to set up variables.

ADDRESS BOOK (of actors)

```
{…
{short_name :  {
                    'address_type':  actor_type,
                    'address': <thespian.actors.ActorAddress>   #Actor's address
                    'component_class':  actor_class_loc,
                    'component_number': 1,  #instance number of the Actor
                    'short_name':  short_name
                    },
…}
self.address_book.get_addresses()   # get address book dictionary
self.address_book.merge_addresses(addresses)  # merge addresses into actor’s address book
self.address_book.get_agents()  # get agent type addresses
self.address_book.get_institutions()  # get institution type addresses
self.address_book.num_agents()  # number of agent type addresses in address book
self.address_book.num_institutions()  # number of institution type addresses in address book
self.address_book.select_addresses(selector)  # Returns list or single address
self.address_book.broadcast_message(selector, message)  # Send message to all selected
        selector = {"address_type": "agent"}
                 = {"address_type": "institution"}
                 = {"short_name":  short_name}
```