- This is the system entry point for the Operators
- It allows the Operator to start/stop the Scheduler's automatic mode, and provides them manual access to the low level components
- It should provide a component that implements all methods, and a TUI client to access those methods.

Methods

- Generic Methods
  - void setMode(in boolean mode): Interacts with the **Scheduler** component in the following way:
    - If internal mode is changed from **Manual** to **Manual** nothing happens
    - If internal mode is changed from **Manual** to **Automatic**, **start** method in **Scheduler** gets called
    - If internal mode is changed from **Automatic** to **Manual**, **stop** method in **Scheduler** gets called
    - If internal mode is changed from **Automatic** to **Automatic**, **SYSTEMErr::AlreadyInAutomaticEx** exception is raised
  - boolean getMode():
    - Returns the current value of the internal mode (Either **Manual** or **Automatic**)
  - Initial value for internal mode should be **Manual**
- Instrument Interaction
  - void cameraOn(): Method **cameraOn** on **Instrument** component gets called
  - void cameraOff(): Method **cameraOff** on **Instrument** component gets called
  - TYPES::ImageType getCameraImage(): Method **takeImage** on **Instrument** component gets called, returning the image
  - void setRGB(in TYPES::RGB rgbConfig): Method **setRGB** on **Instrument** component gets called, passing arguments along
  - void setPixelBias(in long bias): Method **setPixelBias** on **Instrument** component gets called, passing arguments along
  - void setResetLevel(in long resetLevel): Method **setResetLevel** on **Instrument** component gets called, passing arguments along
- Telescope Interaction
  - void moveTelescope(in TYPES::Position coordinates): Method **moveTelescope** on **Telescope** component gets called, passing along the telescope coordinates
  - TYPES::Position getTelescopePosition(): Method **getTelescopePosition** on **Telescope** component gets called, returning the current telescope position

Dummy Components

- For development testing, you require the following simulated components:
  - Scheduler
  - Instrument
  - Telescope

- This is the system entry point for the Astronomers
- Besides allowing an astronomer to:
  - Store a target list
  - Query for the status
  - Retrieve the proposal observations
  - It provides methods to get the proposals currently inserted into the Database
  - To set a status to a given proposal
  - To insert a given observation into the Database
- A *Proposal* consists of:
  - *TargetList* (which is a list of one or more *Target*)
  - Identifier and a status (0 - queued, 1 - running, 2 - ready)
  - A unique identifier is assigned by the database component and returned to the client after storing its *TargetList*
- A *Target* consists of:
  - *Position* specification
  - Exposure time
  - Target identifier assigned by the astronomer
    - The target identifiers should be unique per proposal
- A *Position* is simply the telescope position to be reached for that observation

Methods

- long storeProposal (in TYPES::TargetList targets):
  - Stores the given **TYPES::TargetList** in some kind of database (Memory, SQLite, Redis, etc.), along with the **proposal status** and the **proposal ID**
    - Proposals are initiated with **proposal status** with **STATUS_INITIAL_PROPOSAL** value
    - It returns a unique **porposal ID** for the proposal
    - The **proposal ID** has to be unique
- long getProposalStatus(in long pid):
  - Returns the **proposal status** for the given **proposal ID**
  - In case the **proposal ID** is not recognized, it returns **STATUS_NO_SUCH_PROPOSAL**
- void removeProposal(in long pid):
  - Removes the proposal associated with given **proposal ID**
  - If the **proposal ID** is not present, then do not execute any operation and don't report any problem
- TYPES::ImageList getProposalObservations(in long pid):
  - Retuns all images associated to a given **proposal ID**
  - Throw **SYSTEMErr::ProposalNotYetReadyEx** exception if **proposal ID** has not been executed yet
- ProposalList getProposals():
  - Returns stored proposals which have not been executed yet
    - Proposals with queued status
    - If there are no pending proposals returns an empty list
- void setProposalStatus(in long pid, in long status):
  - Set the proposal status
  - Raises **SYSTEMErr::InvalidProposalStatusTransitionEx** if the change is not from queued(0) to running(1) or from running(1) to ready (2)
- void storeImage(in long pid, in long tid, in TYPES::ImageType image):
  - Stores an image for a given **proposal ID** and **target ID**
  - Raises **SYSTEMErr::ImageAlreadyStoredEx** if an image has already been stored for the given **target ID** and **proposal ID** combination

Dummy Components

- No other components are needed to test Database Component

- The Scheduler is responsible to:
  - Select a proposal from the Database
  - Execute a proposal
  - Store the observations
  - Manage the proposal's lifecycle
- The observations are scheduled automatically according to some scheduling algorithm as soon as the scheduler is requested to start
- On stop it will complete the executing proposal before suspending the automatic mode

Methods

- void start():
  - Starts the **Scheduler**
  - The **Scheduler** will loop through all available **proposals**, either until all **proposals** are done or until the **stop** method is called
  - If the **Scheduler** is already running, then throws **SYSTEMErr::SchedulerAlreadyRunningEx**
- void stop():
  - Stops the **Scheduler**
  - This will stop the **Scheduler** from scheduling more **proposals**
  - It will **not(!)** break the ongoing **observation**, and will return only when the running **observation** has finished
  - If the Scheduler is not running, then throw **SYSTEMErr::SchedulerAlreadyStoppedEx**
- long proposalUnderExecution():
  - Returns the **proposal id** of the proposal currently under execution
  - Raises **SYSTEMErr::NoProposalExecutingEx** if no proposal is executing.

Dummy Components

- For development testing, you require the following simulated components:
  - Database
  - Instrument
  - Telescope

- This component communicates with the low level hardware access layer to:
  - Execute observations; i.e. moves the telescope to a given position
  - Acquires the image from the Instrument for a given exposure time once the telescope is in position

Methods

- TYPES::ImageType observe(in TYPES::Position coordinates, in long exposureTime): Calls the internal method **moveTo** and then makes use of **Instrument** method **takeImage** passing along exposure time in milliseconds.
- void moveTo(in TYPES::Position coordinates): Commands the **Telescope** to move to the commanded position by calling **TelescopeControl** method **objfix**
- TYPES::Position getCurrentPosition(): Returns the current **Telescope** position

Dummy Components

For development testing, you require the following simulated components:

- Instrument
- TelescopeControl

- Sets the CCD camera on and off
- Takes an image with a given exposure time

Methods

- void cameraOn(): Method **on** on **Camera** component gets called
- void cameraOff(): Method **off** on **Camera** component gets called
- TYPES::ImageType takeImage(in long exposureTime): Method **getFrame** on **Camera** component gets called, passing along shutter speed value and setting a default iso value, returning the image
    - Default iso value is iso100
    - Shutter speed value depends on exposureTime value. The exposureTime parameter is passed as milliseconds. Check for the following conditions to determine the shutter speed value:
        - If exposure time in seconds is less or equal to 1/5, shutter speed is s1over5.
        - If exposure time in seconds is less or equal to 1/4, shutter speed is s1over4.
        - If exposure time in seconds is less or equal to 0.3, shutter speed is s0_3.
        - If exposure time in seconds is less or equal to 0.4, shutter speed is s0_4.
        - If none of the conditions above is met, shutter speed is s1.
- void setRGB(in TYPES::RGB rgbConfig): This is just a placeholder method for better **Camera** implementations

    - Store value in local vairable
- void setPixelBias(in long bias): This is just a placeholder method for better **Camera** implementations
    - Store value in local vairable
- void setResetLevel(in long resetLevel): This is just a placeholder method for better **Camera** implementations
    - Store value in local vairable

Dummy Components

For development testing, you require the following simulated components:

- Camera