

Simulation activity in CCS

Objectives

The simulation activity in CCS must provide software tools to test CCS code before the mechanical subsystems exist.

Current status

A first simulator has been created for the FCS. It has been designed to be reused by other subsystems if needed.

The principles is to build a simulator from modular mechanical items which can be :

- mobil mechanical item
- engine
- latch

Some constraints can be defined between this mechanical items. For example :

- a latch can lock a mobil mechanical item
- an engine can move a mobil mechanical item

A modular simulator is described in a XML file.

Use Cases for a Filter Changer modular simulator

See the attached files below for the complete description of the use cases.

We have defined 4 use cases :

- Move carousel to position X
- Stop carousel
- Lock latch
- Unlock latch

How to interface a modular simulator with CCS

To interface a modular simulator with the CCS code, we have to define a simulation modular subsystem in the CCS. This simulation subsystem is composed of a single module. See below the FCS simulator as an example.

The simulation subsystem is in the package : `ccs/simulation`. Its single module is the `SimuDriverModule`. The `SimuDriverModule` class extends the `ccs/localbus/Module` class.

Description of the `SimuDriverModule` class

See javadoc for more details.

Here are the main methods to implement in the `SimuDriverModule` class:

- Overwritten methods from the super Class `Module`
 - ◆ `initModule` : after the `Module` initialization, we want to initialize the modular simulator from its XML file.
 - ◆ `tick` : what we do for each tick of the timer : notify observers, publish on the log bus the new values of the data we update.
- One method for each command we want CCS send to our simulator module (See more details in Use cases below)
 - ◆ `Move` : move carousel in the position given as a parameter
 - ◆ `Stop` : stop carousel
 - ◆ `Lock` : lock the latch given as a parameter
 - ◆ `Unlock` : unlock the latch given as a parameter
- Getter and setter methods

FCS simulator USER GUIDE

We have defined a very simple FCS simulator with an engine, a latch (`latchA`) and a carousel. We can drive this simulator from the CCS as we do with any other subsystem.

First, download and compile the current code

Follow the instructions in [CCS code short description](#).

Then, test the simulator :

You have to open 3 windows.

1. window 1: run the `jmsserver` :

Read the instructions in [CCS code short description](#).

2. window 2: run the simulation sub-system :

```
cd LSSTDIR/trunk/CameraControl/  
./run-subsys conf/test-simulator.xml
```

The file `test-simulator.xml` contains the description of our simulation subsystem. It is very simple as our simulator is very simple :

```
<?xml version="1.0" encoding="UTF-8"?>  
<modules name="test-simu">  
  
<module name="FCS-Simulator" class="org.lsst.ccs.simulation.SimuDriverModule">  
  <property name="simulatorConfigFile" value="conf/modularSimulator-toto.xml"/>  
  
</module>  
  
</modules>
```

The 3 important things in this file are :

- the name of the modular subsystem : `test-simu`

You have to know it to be able to "talk" with it from the TestConsole.

- the name of the module which is the interface with the simulator : `FCS-Simulator`

You have to know it to be able to "talk" with it from the TestConsole.

- the name of the XML file where is described our modular simulator :

`conf/modularSimulator-toto.xml`

3. window 3: run a Test Console to send commands to the Simulation sub-system :

```
cd LSSTDIR/trunk/CameraControl/  
./run-module org.lsst.ccs.test.TestConsole
```

TestConsole is a BusMaster. You are now ready to send commands to the Simulation subsystem. For example, type in the TestConsole:

```
invoke test-simu/FCS-Simulator move 15.00  
invoke test-simu/FCS-Simulator move 350.00  
invoke test-simu/FCS-Simulator lock latchA  
invoke test-simu/FCS-Simulator move 25.00  
invoke test-simu/FCS-Simulator unlock latchA  
invoke test-simu/FCS-Simulator move 25.00
```

and read what happens in the window 2.

Then, test the simulator :