PAUL SCHERRER INSTITUT

PSI

Introduction to ooEpics

Controls Talk, PSI
Zheqiao Geng
05.09.2016

PAUL SCHERRER INSTITUT

Outline

- Motivation
- InternalData – a general device support
- Concept of software architecture behind ooEpics
- C++ classes of ooEpics
- Code generation tool

PAUL SCHERRER INSTITUT

Motivation

Difficulties when developing EPICS software …

- A clear understanding how record processing works is required to design new device support routines
- It is difficult to implement functions taking data from and putting results to many PVs
- Editing template files is labor intensive
- Links in database contain lots of algorithms and logics which is not so easy to read and understand
- …

Just personal wishes

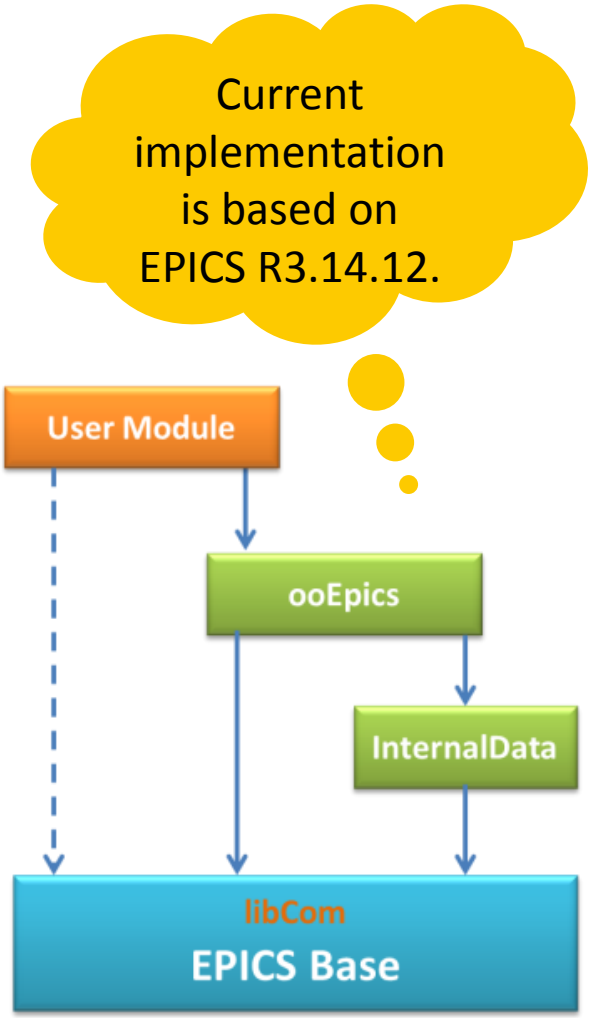
I need something to

- Allow developing EPICS software without knowing much about EPICS concepts (e.g. record/device/driver supports)
- Easily get or set values of multiple PVs in a function executed by a private thread
- Possibly be able to automatically generate EPICS database and other configuration files (e.g. autosave request)
- Implement a general architecture which should fit to different applications

These led to ooEpics …

What is ooEpics ?

- ooEpics stands for “Object Oriented EPICS framework”. It is a C++ framework designed for EPICS module development. ooEpics contains C++ classes with the following categories:
  - Classes to manage and configure user EPICS modules
  - Base classes to derive the key components for user EPICS modules
  - Wrapper classes for EPICS data access and Channel Access client
- ooEpics classes are packed into an EPICS module and it depends on another module called InternalData
- Both the ooEpics module and InternalData module are implemented based on standard C/C++ libraries and EPICS platform independent libraries (libCom)



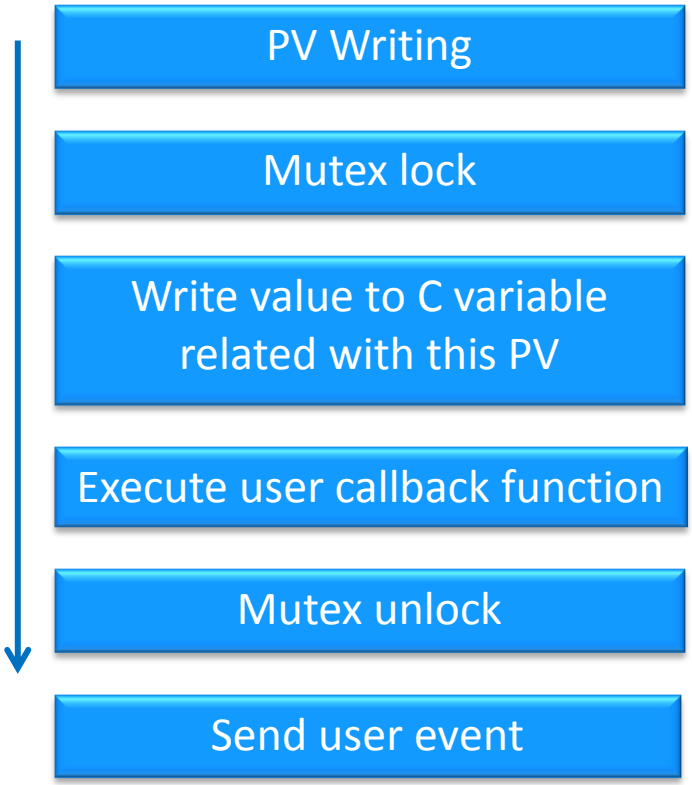
PAUL SCHERRER INSTITUT

InternalData – a general device support

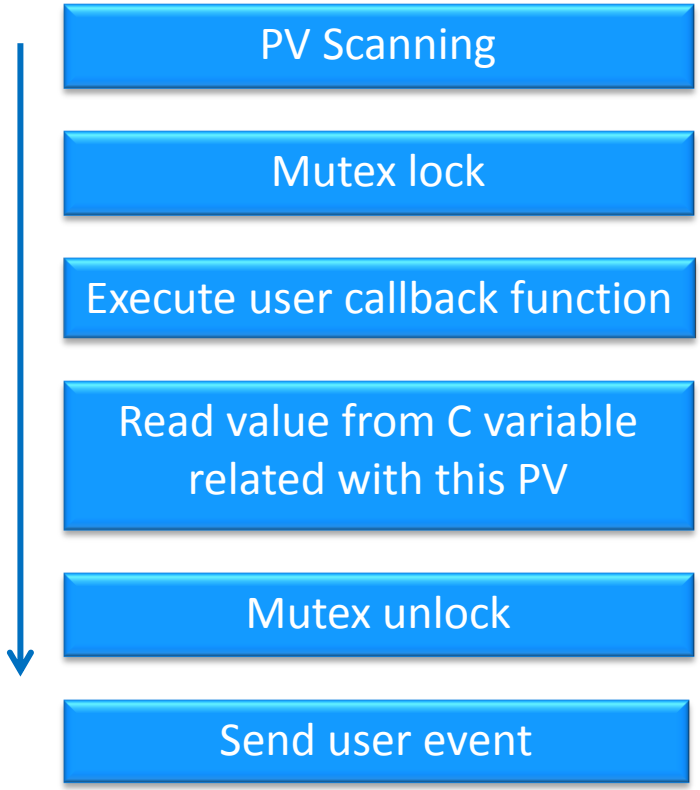
InternalData processes

- In user code, an InternalData node can be created to link a C variable with an EPICS record with optional mutex, callback function and event
- Each InternalData node will be represented by an EPICS record

Procedure to process an output PV (e.g. ao, bo) with InternalData

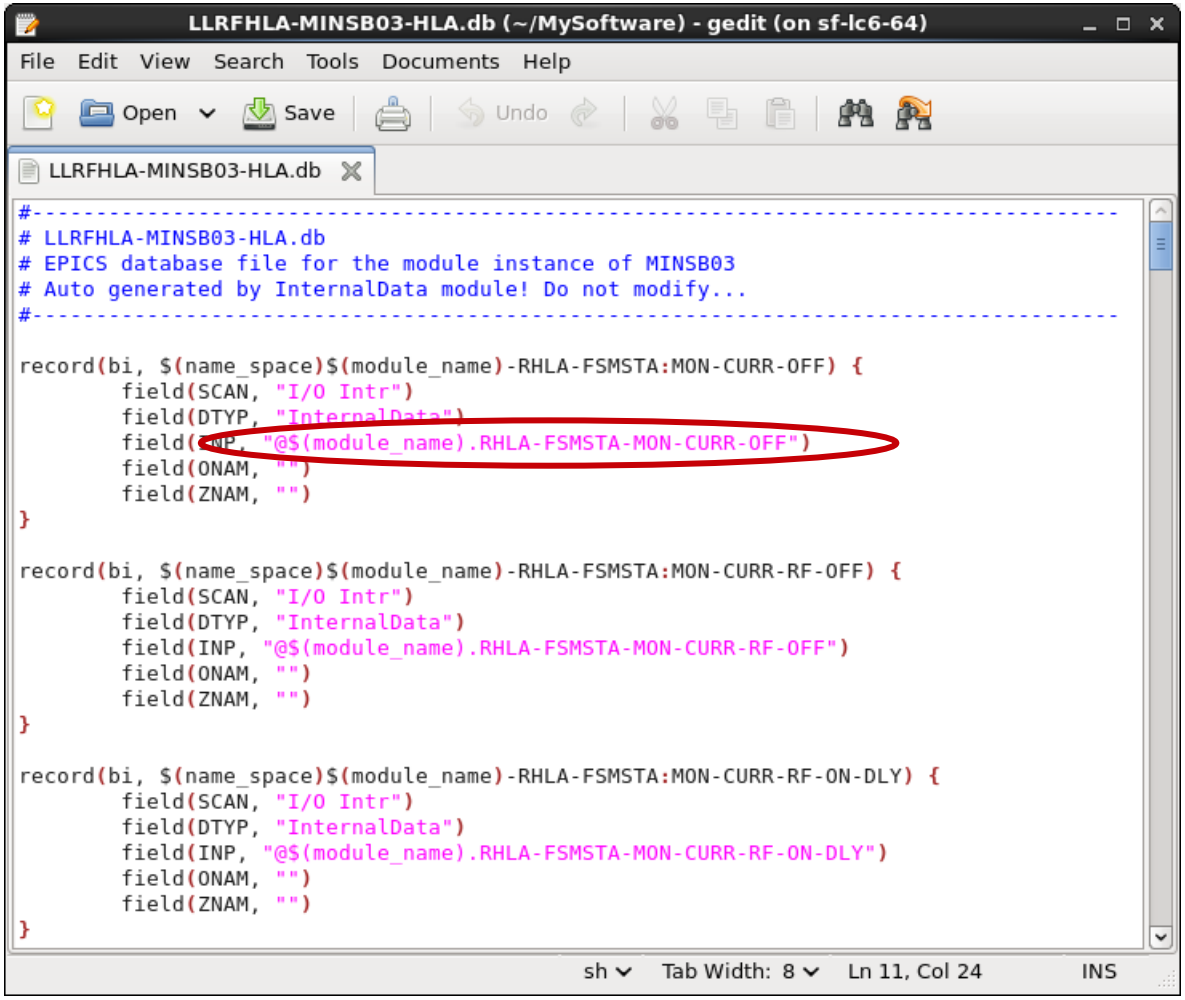


Procedure to process an input PV (e.g. ai, bi) with InternalData



InternalData tools

- Generation of EPICS database, autosave request file and archiver configuration file



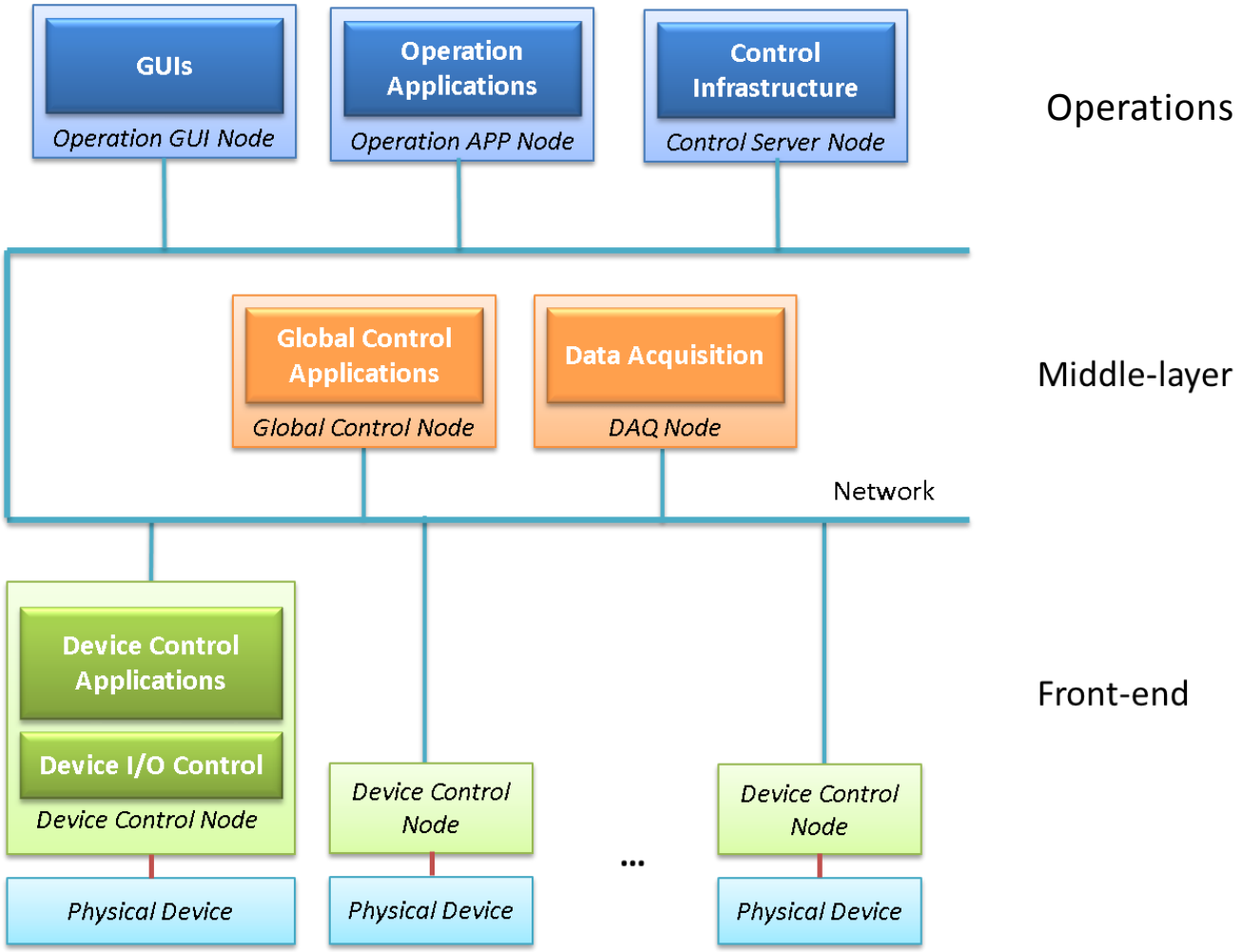
Manually editing the template file is fully supported!

The strings defined in the C/C++ code should be passed via links to build associations between records and variables.

PAUL SCHERRER INSTITUT

Concept of software architecture behind ooEpics

EPICS based control system



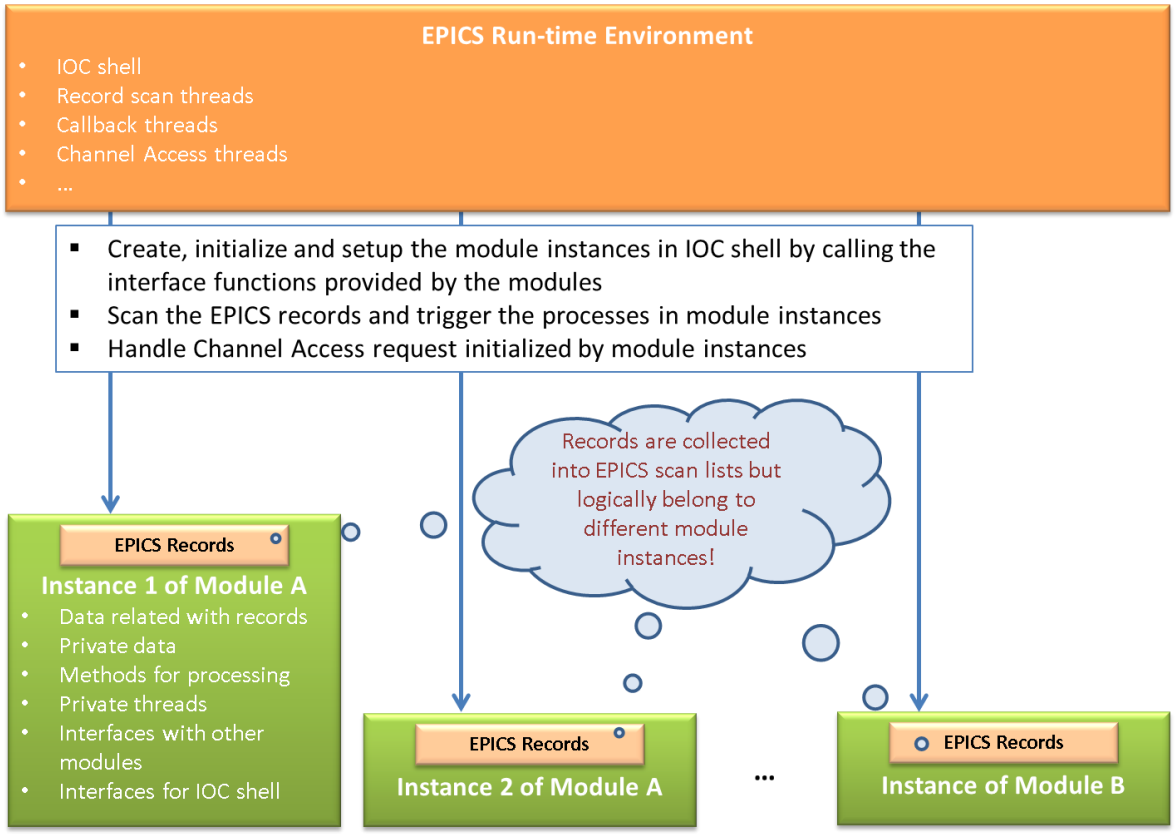
The ooEpics framework will focus on the development of front-end and middle-layer components.

General architecture of EPICS software

The user codes can be organized into different modules based on the group of functions.

A module can contain the following items:

- EPICS records to configure the parameters of the module and display the results
- Function blocks to process the data and perform logical controls
- Maybe a thread to actively do its job
- Interfaces with other modules
- Interfaces for IOC shell to create, initialize or setup the module instances



Types of EPICS modules

❑ *Control Device Module*

The “Control Device” is the instrument inserted by the control system designer to control a physical device. For example, klystron is a physical device which is essential to the RF system, but the FPGA board in LLRF system is viewed as a control device.

The basic functions that a control device module should provide include opening device, closing device, setting/reading registers and transferring block data with DMA.

❑ *Domain Device Module*

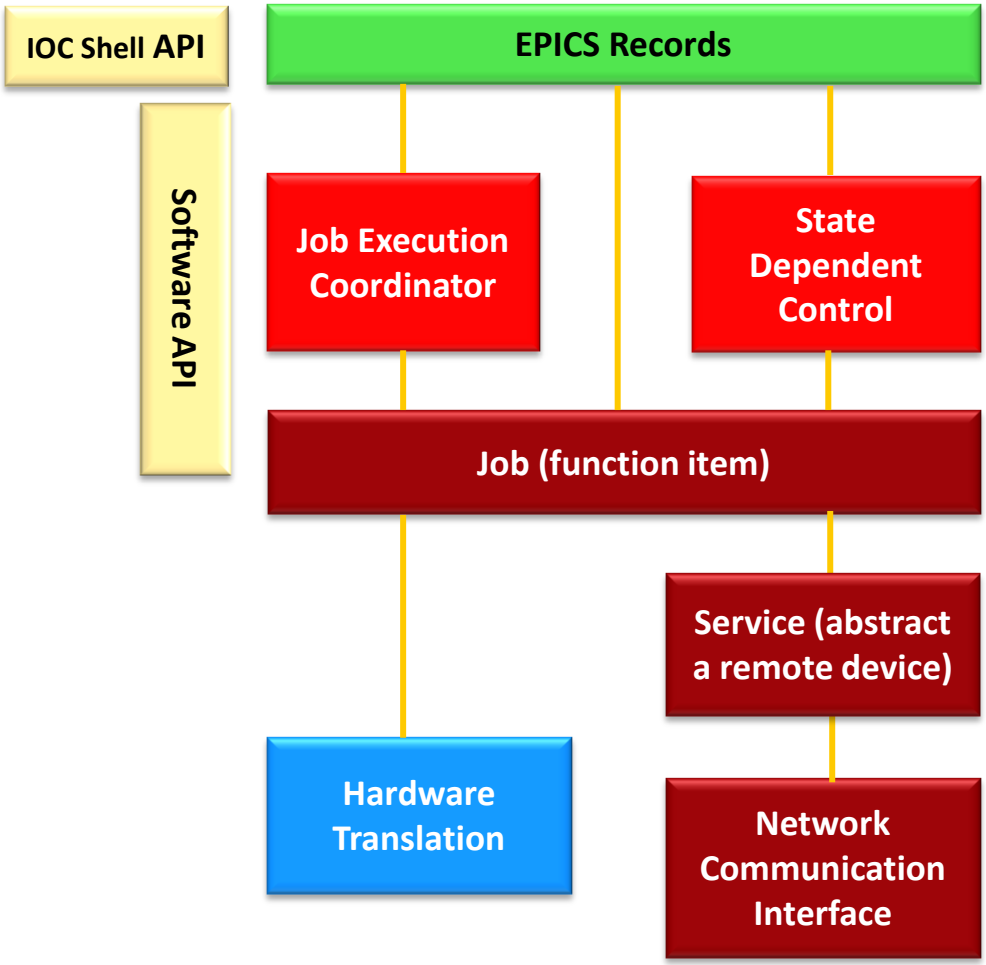
A “Domain Device” is the physical device essential for the function of a system (e.g. klystron). The module to control a domain device uses the functions provided by control device modules and interprets the data as specific physical quantities. For example, the domain device module to control a klystron will get the ADC data and interprets it as an RF signal waveform.

The domain device module also implements some complex functions for data processing and control (e.g. pulse-to-pulse feedback for a klystron).

❑ *Application Module*

An “Application Module” implements functions to handle data from domain device modules and perform other middle layer functions. The applications do not directly interact with hardware but use flexible interfaces to communicate with other modules (e.g. Channel Access).

Typical architecture of EPICS modules

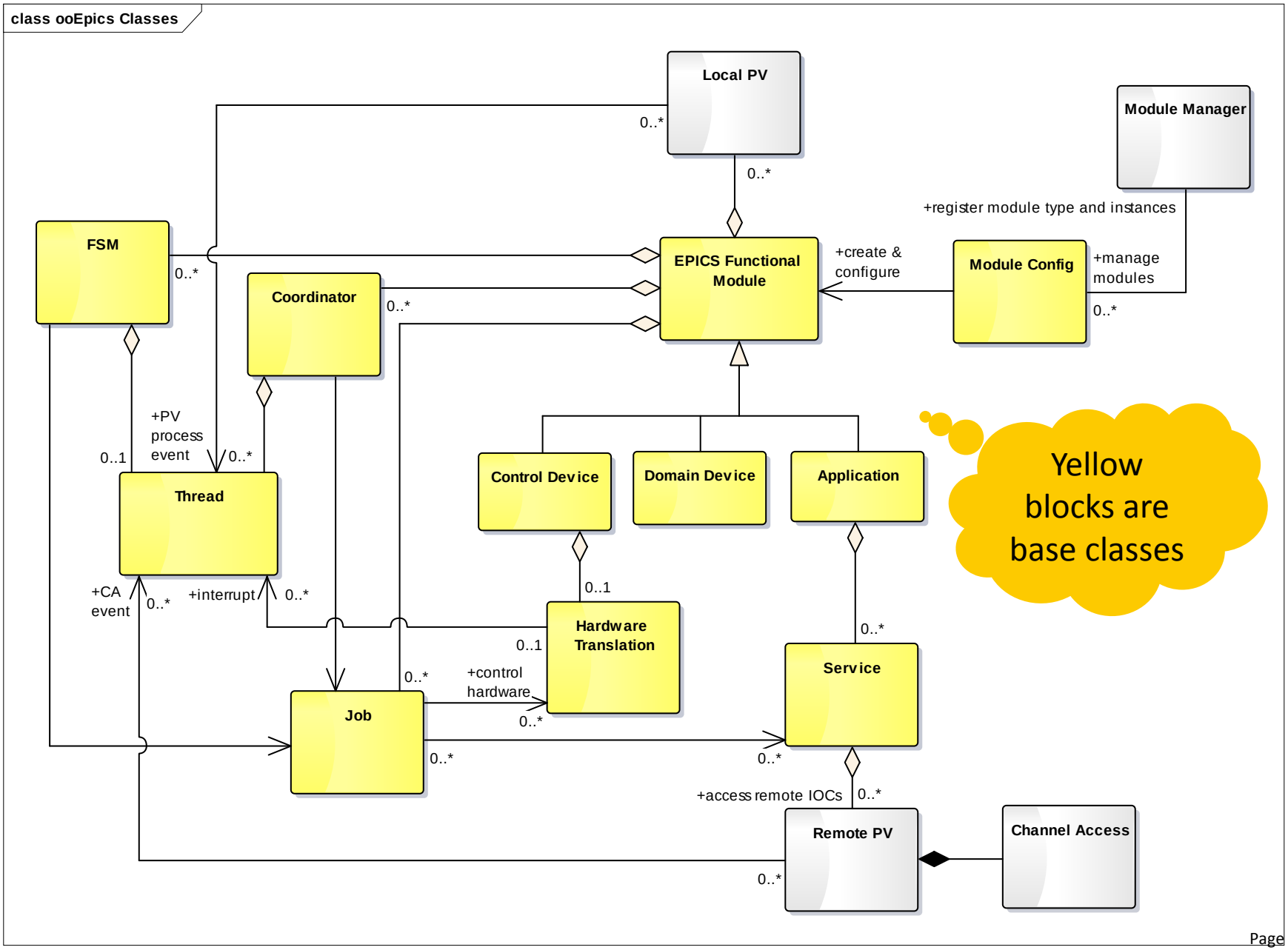


- The “IOC Shell API” is required for a module to be configured in IOC shell
- The “Software API” is optional which is mostly used in a Control Device Module
- The “Job Execution Coordinator” and “State Dependent Control” are active objects mostly driven by threads
- The “Job” is an abstraction of a function item that need to be executed (e.g. write a register, a procedure to calibrate the beam energy)
- A “Service” is an abstraction of an external device which needs to be monitored or controlled in the module. It is mostly used in an Application Module
- The “Network Communication Interface” is mostly used in an Application Module in the middle-layer nodes (e.g. Channel Access client)
- The “Hardware Translation” is mostly used in a Control Device Module to wrap the hardware driver of the device

PAUL SCHERRER INSTITUT

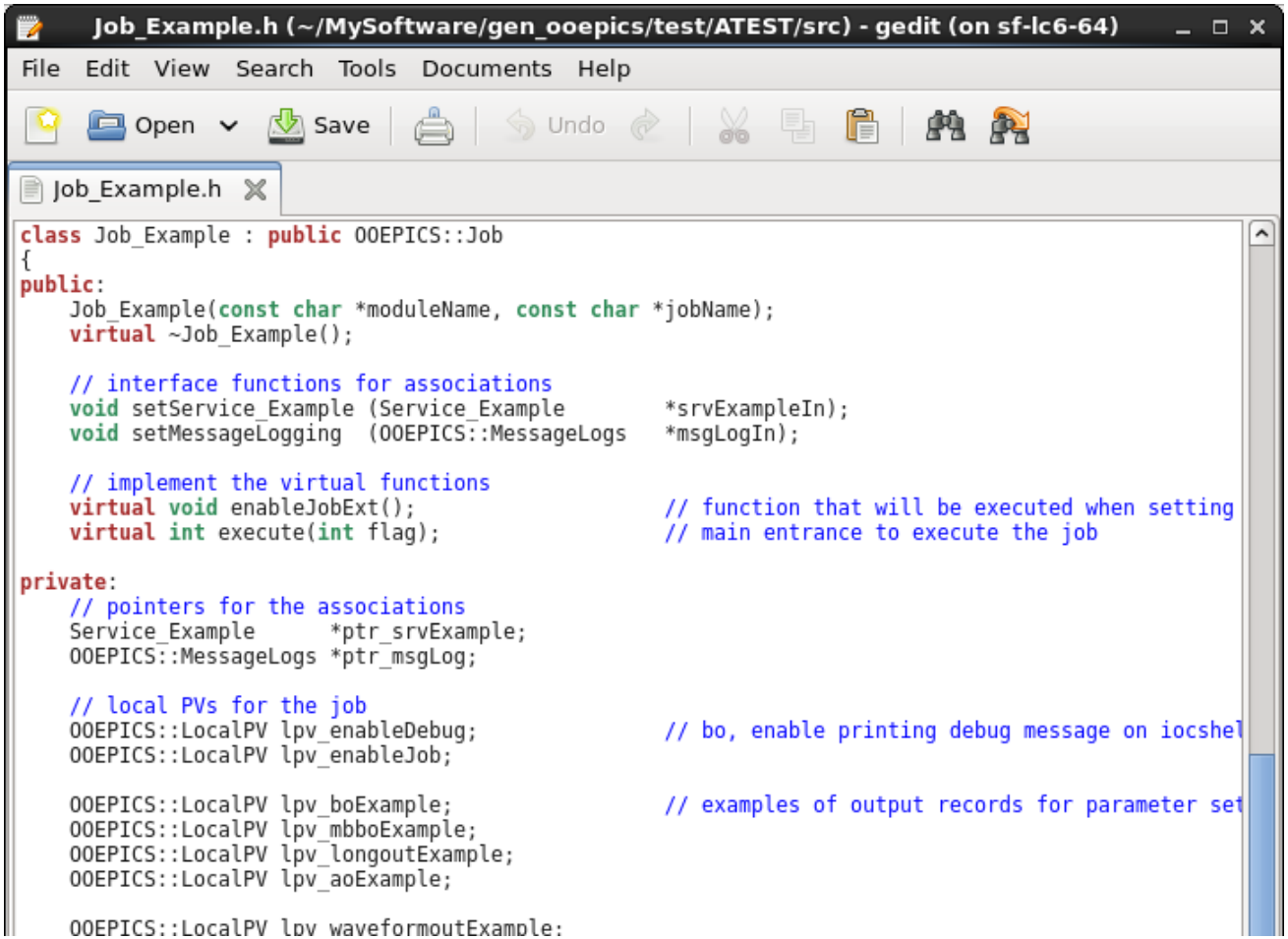
C++ classes of ooEpics

Classes of ooEpics framework



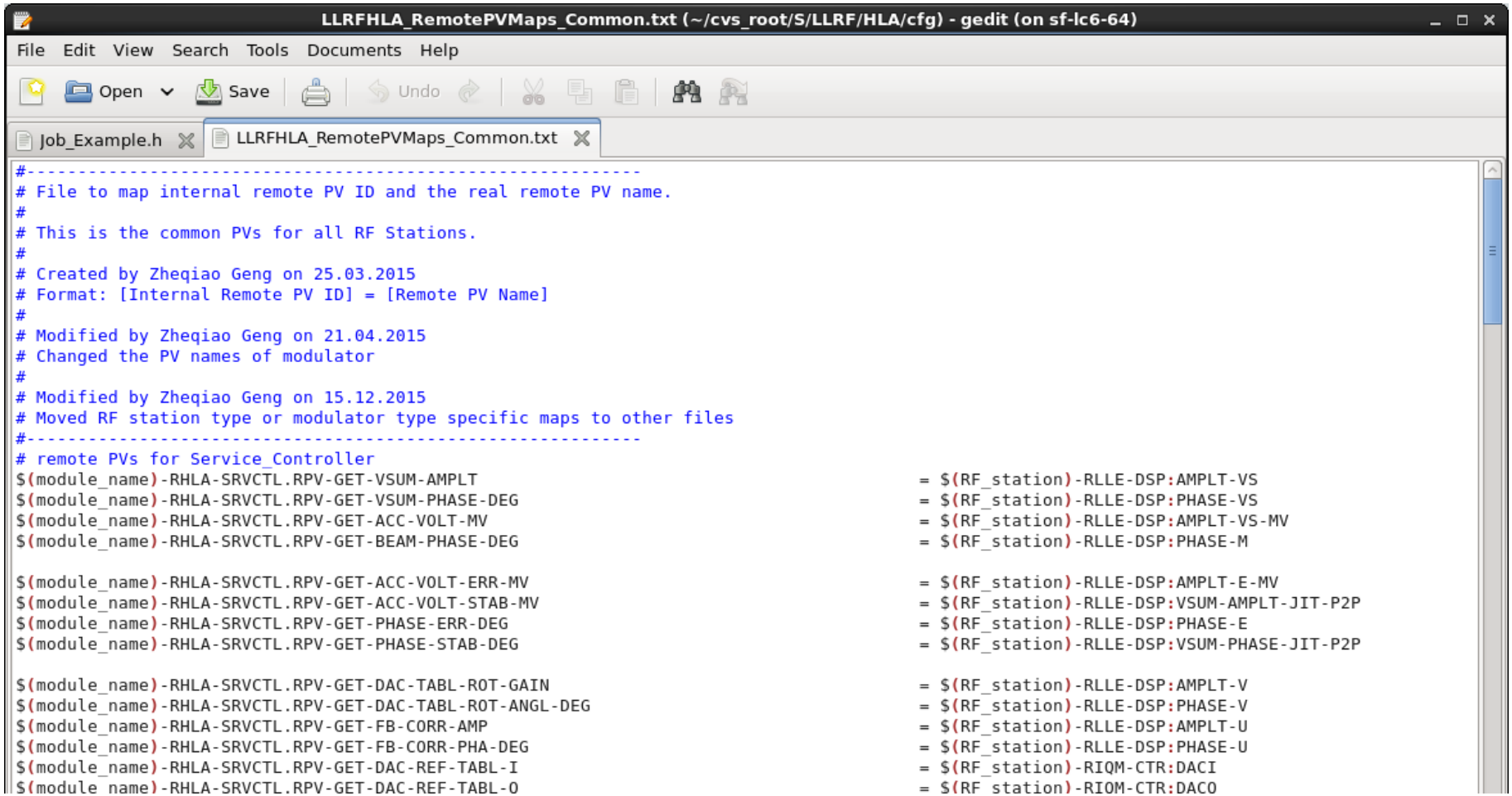
Some key features in ooEpics

EPICS database can be created by creating objects of LocalPV in user code.



Some key features in ooEpics (cont.)

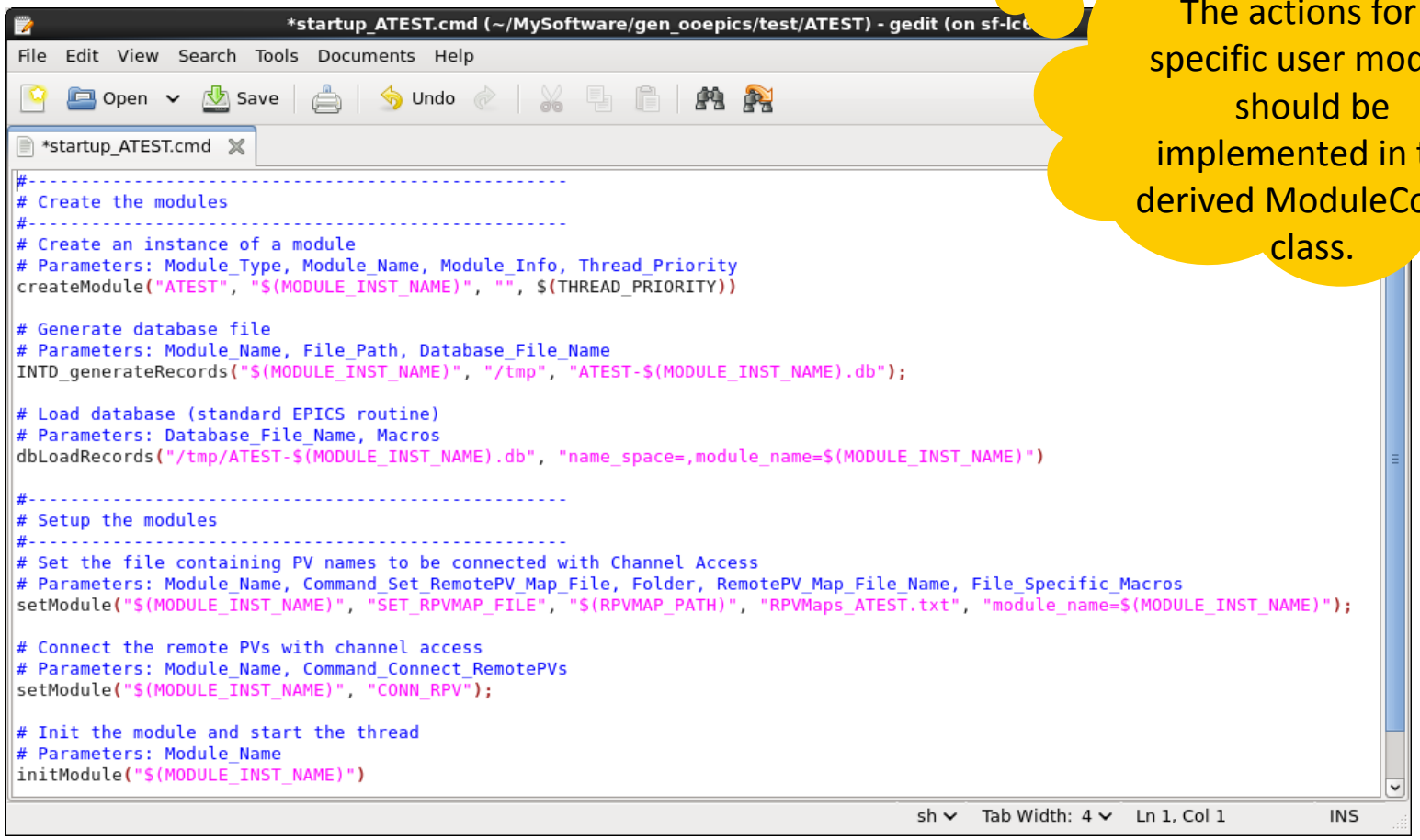
Basic channel access functions (e.g. caget, caput, camonitor, w/o callbacks …) are implemented and wrapped by the RemotePV class. The remote channel names can be managed by a configuration file.



Some key features in ooEpics (cont.)

- Provide base classes to implement finite state machines
- Provide general iocShell functions to create, initialize or setup a module instance:

```
createModule (module_type, module_inst_name, module_inst_info)
setModule    (module_inst_name, command, values[])
initModule   (module_inst_name)
```



The actions for a specific user module should be implemented in the derived ModuleConfig class.

Procedure to develop a model based on ooEpics

1. Derive a child class of one of the module top classes: Control Device, Domain Device or Application. This class will act as a container for all components of the module that you will implement
2. Derive a child class of Module Config for your module and implement the virtual functions to create, register and setup the instance of your module so that the general IOC shell interfaces provided by the Module Manager can be used
3. Based on your design, derive child classes of Coordinator or FSM to have active objects with threads in your module. The child classes of Job, Service or Hardware Translation can be derived to implement specific functions in your module
4. The Local PV objects can be defined in anywhere of your program to define EPICS records for parameter settings or results displaying. Normally the Remote PV objects can be defined in Service classes to access external IOCs with Channel Access

PAUL SCHERRER INSTITUT

Code generation tool

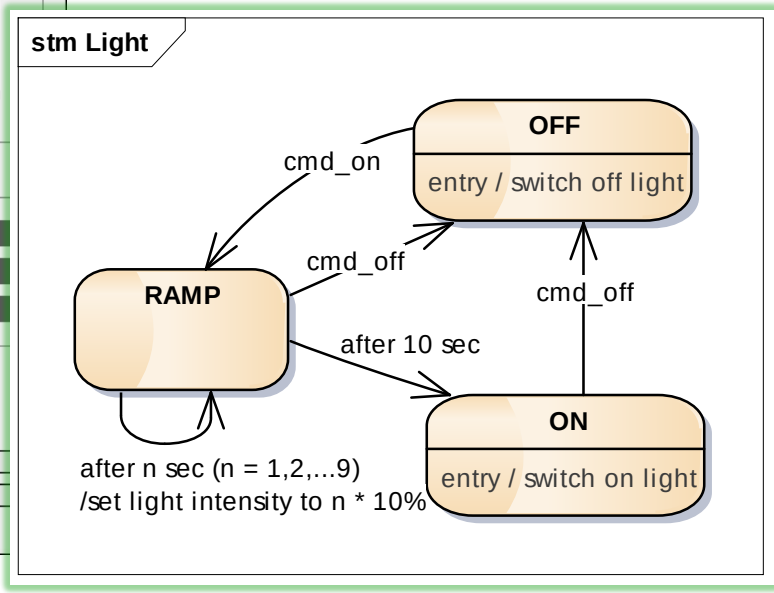
Generate a user module with example codes

```
geng_z@sf-lc6-64:~/MySoftware/gen_ooepics/test (on sf-l
[sf-lc6-64 test]$
[sf-lc6-64 test]$ export OOEPICS_TEMPLATE_PATH=/afs/psi.ch/user/g/geng_z/git_roo
t/ooEpicsGen/template/
[sf-lc6-64 test]$ /afs/psi.ch/user/g/geng_z/git_root/ooEpicsGen/ooEpicsGen.sh AT
EST
-----------------------------------------------------------------
Created framework of ooEpics based module ATEST
-----------------------------------------------------------------
Current folder is /afs/psi.ch/user/g/geng_z/MySoftware/gen_ooepics/test
Generate folder ATEST
Generate folder ATEST/App
Generate folder ATEST/App/config
Generate folder ATEST/App/config/qt
Generate folder ATEST/App/config/archiver
Generate folder ATEST/App/config/alh
Generate folder ATEST/App/config/saveres
Generate folder ATEST/db
Generate folder ATEST/cfg
Generate folder ATEST/src
Generate folder ATEST/test_bench

Current path /afs/psi.ch/user/g/geng_z/MySoftware/ge

Generate in ./src module top class ...
Module_ATEST.h and Module_ATEST.cc generated

Generate in ./src module configure class ...
ModuleConfig_ATEST.h and ModuleConfig_ATEST.cc gener

Generate in ./src job coordinator class ...
Coordinator_Jobs.h and Coordinator_Jobs.cc generate

Generate in ./src example job class ...
Job_Example.h and Job_Example.cc generated

Generate in ./src example service class ...
Service_Example.h and Service_Example.cc generated

Generate in ./src example FSM class ...
FSM_Example.h and FSM_Example.cc generated

Generate in ./src module definitions ...
```

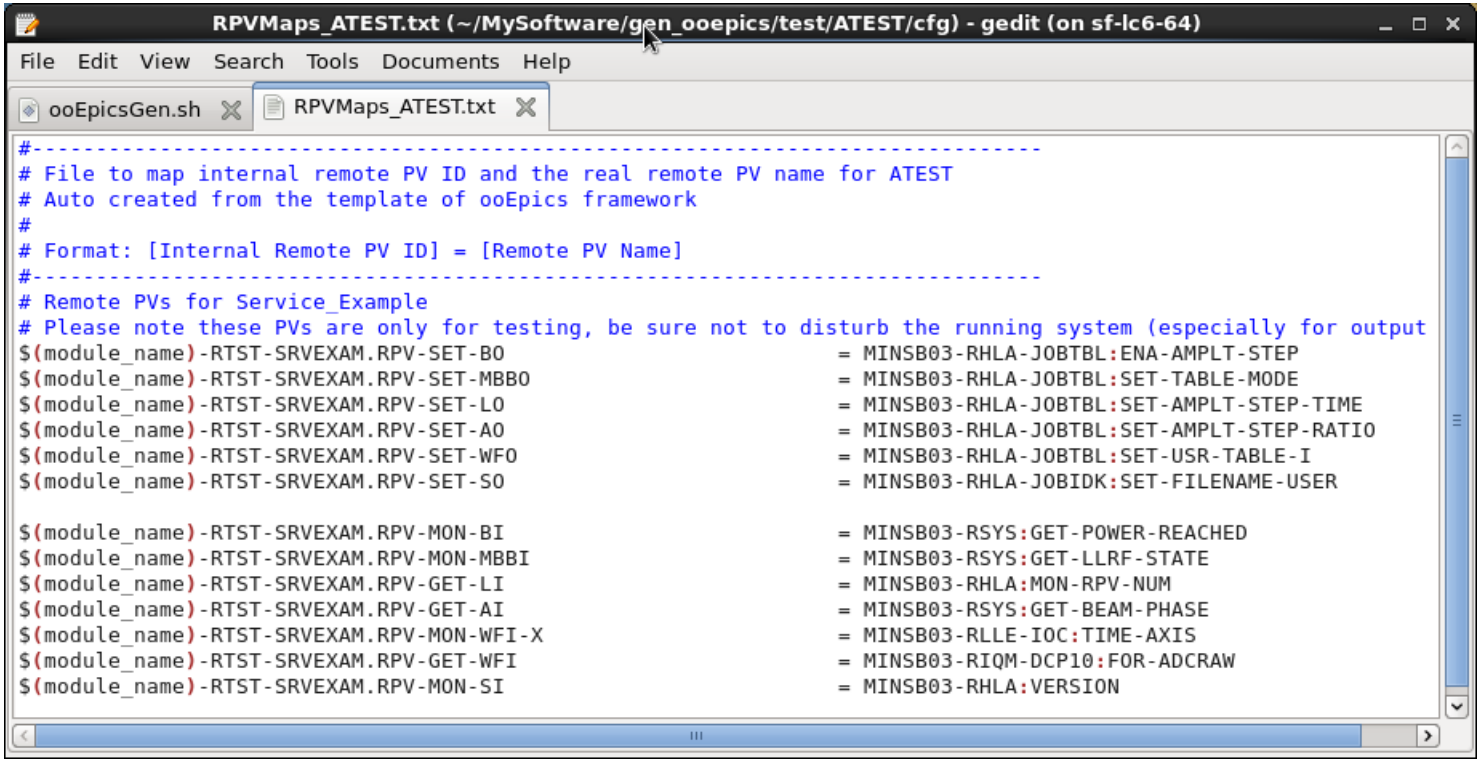
```
geng_z@sf-lc6-64:~/MySoftware/gen_ooepics/test/ATEST (
[sf-lc6-64 test]$ cd ATEST/
[sf-lc6-64 ATEST]$ ll
total 17
drwxr-xr-x 3 geng_z sls 2048 Aug 30 11:52 App
drwxr-xr-x 2 geng_z sls 2048 Aug 30 11:52 cfg
drwxr-xr-x 2 geng_z sls 2048 Aug 30 11:52 db
-rw-r--r-- 1 geng_z sls 2017 Aug 30 11:52 Makefile
-rw-r--r-- 1 geng_z sls  528 Aug 30 11:52 Release_Note
-rw-r--r-- 1 geng_z sls 2009 Aug 30 11:52 saveres_request_gen_ATEST.cmd
drwxr-xr-x 2 geng_z sls 2048 Aug 30 11:52 src
-rw-r--r-- 1 geng_z sls 1885 Aug 30 11:52 startup_ATEST.cmd
drwxr-xr-x 2 geng_z sls 2048 Aug 30 11:52 test_bench
[sf-lc6-64 ATEST]$ ll src/
total 84
-rw-r--r-- 1 geng_z sls  4885 Aug 30 11:52 Coordinator_Jobs.cc
-rw-r--r-- 1 geng_z sls  2883 Aug 30 11:52 Coordinator_Jobs.h
-rw-r--r-- 1 geng_z sls 12689 Aug 30 11:52 FSM_Example.cc
-rw-r--r-- 1 geng_z sls  4537 Aug 30 11:52 FSM_Example.h
-rw-r--r-- 1 geng_z sls 16432 Aug 30 11:52 Job_Example.cc
-rw-r--r-- 1 geng_z sls  3694 Aug 30 11:52 Job_Example.h
-rw-r--r-- 1 geng_z sls  9095 Aug 30 11:52 Module_ATEST.cc
-rw-r--r-- 1 geng_z sls  4470 Aug 30 11:52 Module_ATEST.h
-rw-r--r-- 1 geng_z sls  5188 Aug 30 11:52 ModuleConfig_ATEST.cc
-rw-r--r-- 1 geng_z sls  1165 Aug 30 11:52 ModuleConfig_ATEST.h
-rw-r--r-- 1 geng_z sls  1323 Aug 30 11:52 ModuleDefs_ATEST.h
-rw-r--r-- 1 geng_z sls  9156 Aug 30 11:52 Service_Example.cc
-rw-r--r-- 1 geng_z sls  3464 Aug 30 11:52 Service_Example.h
[sf-lc6-64 ATEST]$
```

Test GUI for example codes

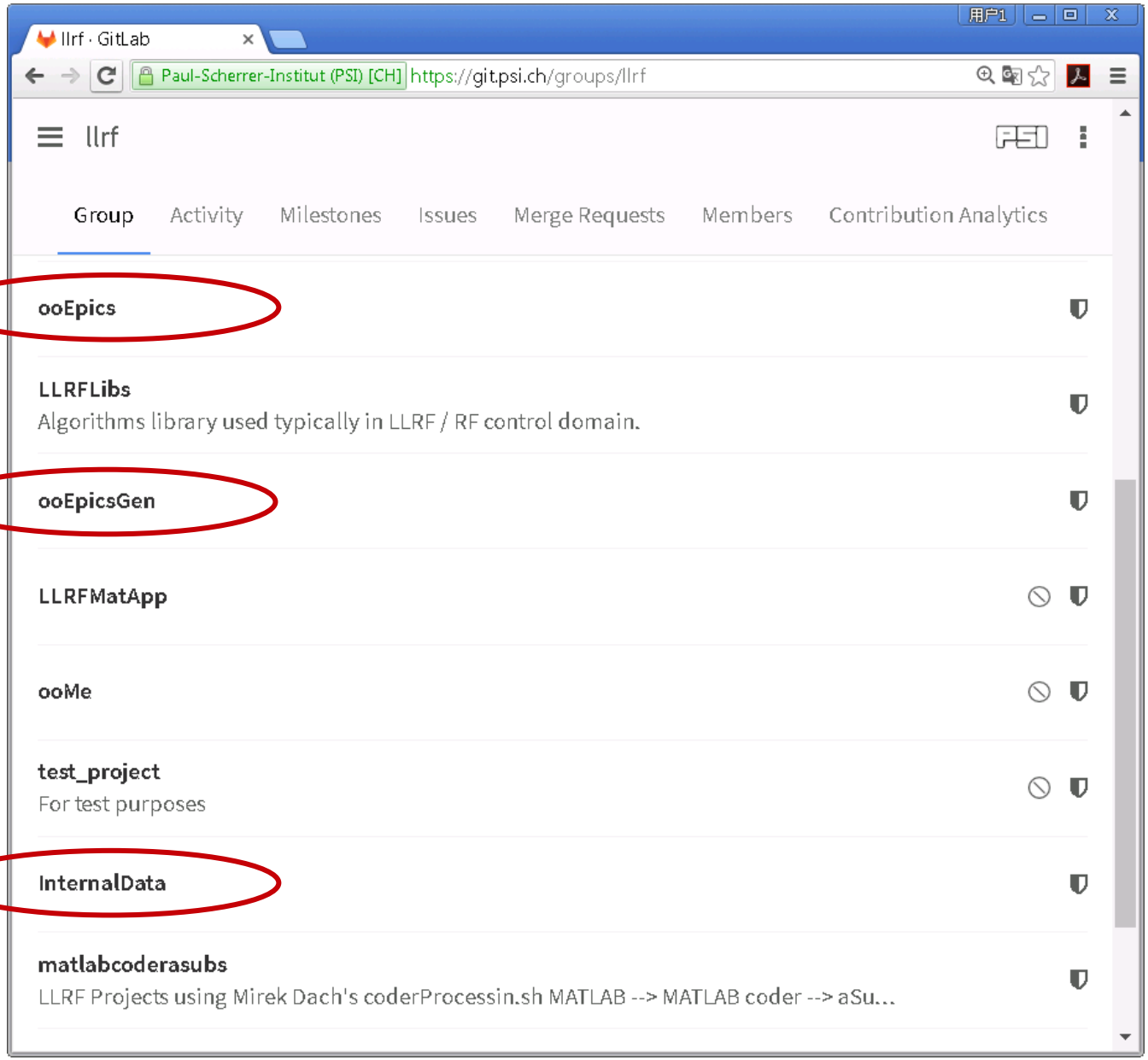
- Use a coordinator to execute the example job to get or set remote PV data via the example service
- An example FSM to ramp the light



Remote PV names for example codes



Source codes in Git



PAUL SCHERRER INSTITUT

PSI

Questions?