Sardana & Continuous Scans

Zbigniew Reszela

on behalf of Sardana Community

Continuous Scans Workshop
20-21/09/2023

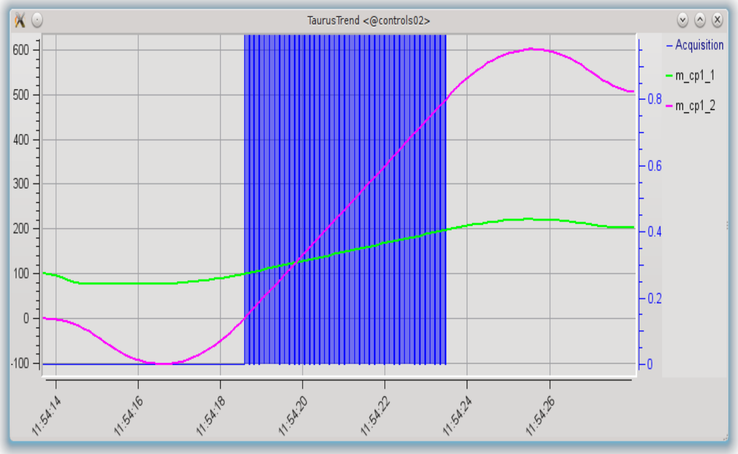


Table of contents

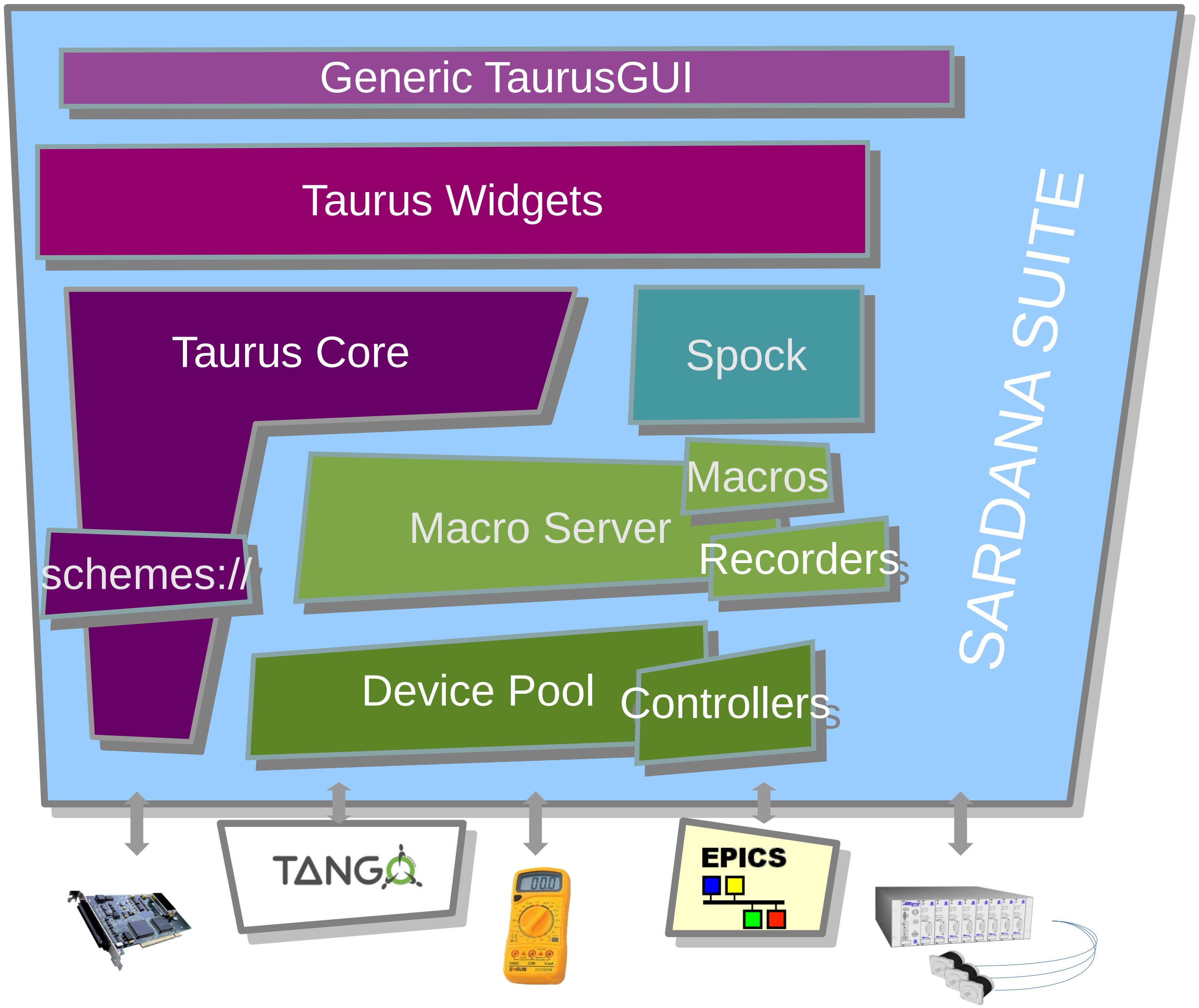
Introduction to Sardana

Continuous Scan Approach

Continuous Scan Details

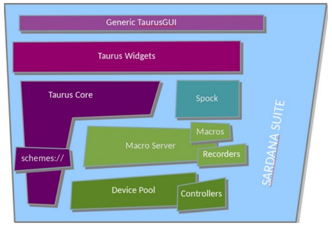
Status & Roadmap

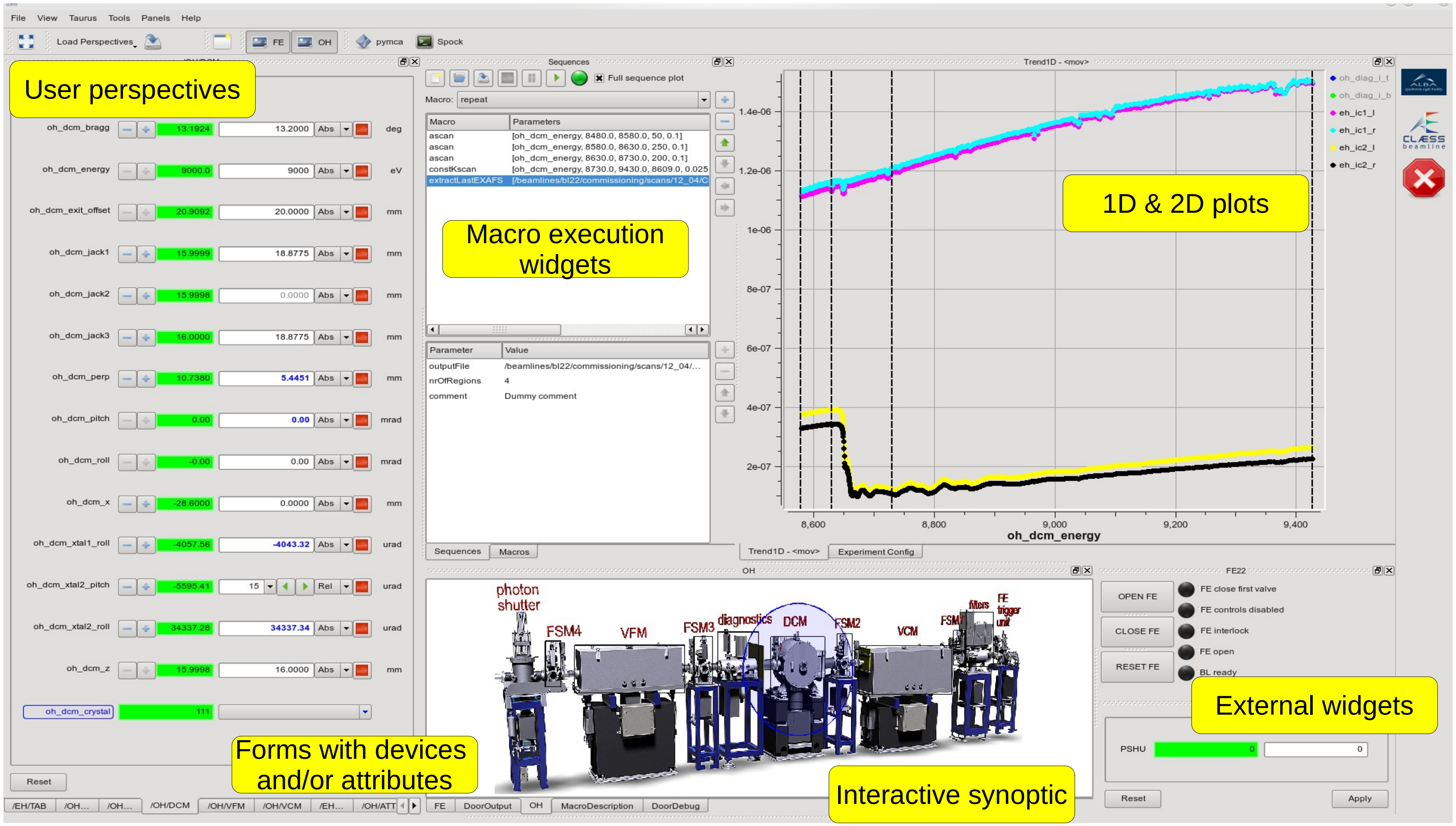
Overview



Generic TaurusGUI
Taurus Widgets
Taurus Core
Spock
Macros
Macro Server
Recorders
schemes://
Device Pool
Controllers
SARDANA SUITE
TANGO
EPICS

Generic Taurus GUI

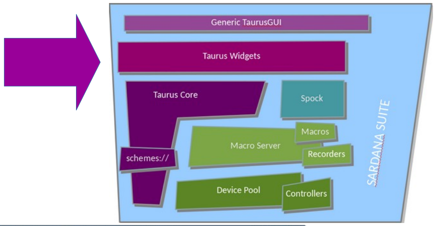


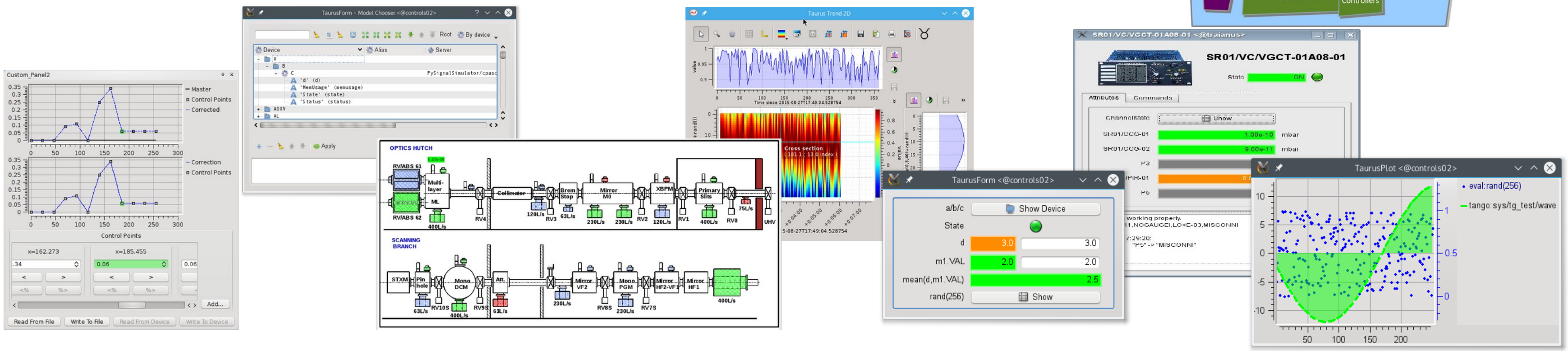


BL22 (ALBA) GUI created with the TaurusGUI framework

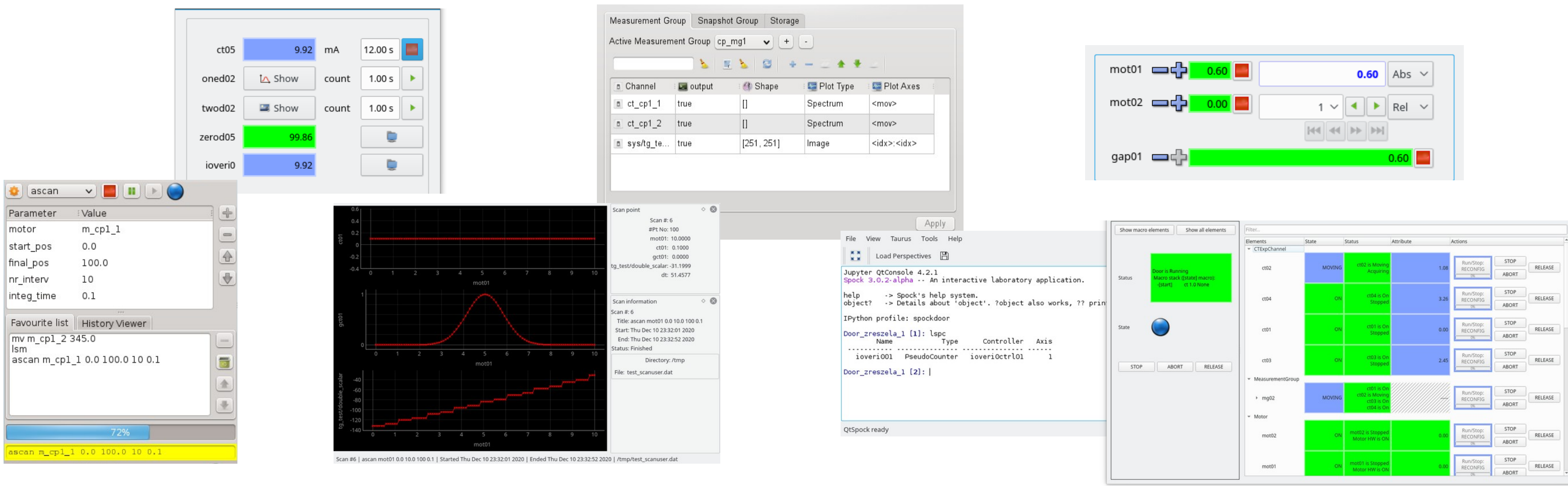
TaurusGUI framework for creating complete GUIs without programming a single line of code

Taurus Widgets

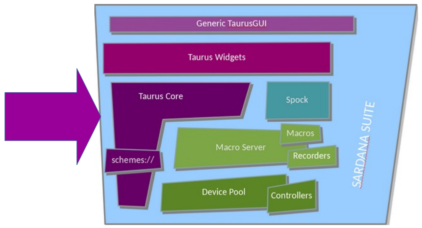




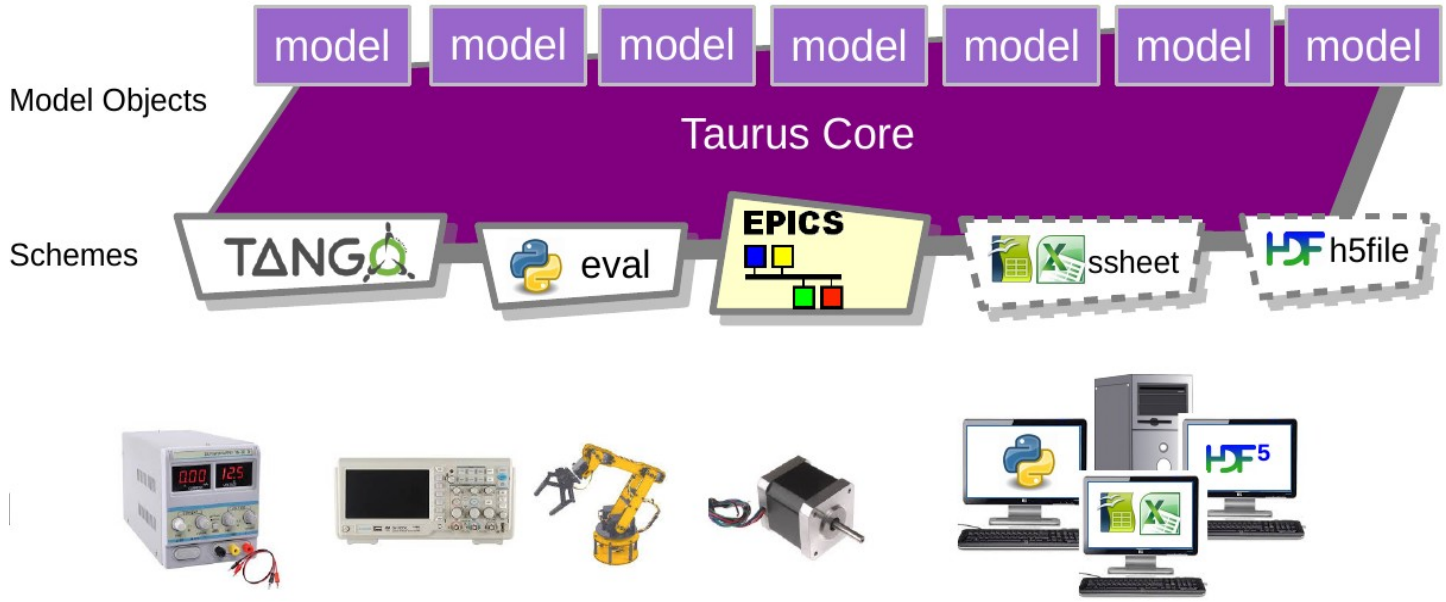
- Qt based graphical widgets: generic forms, plots, ...
- Sardana specific widgets: macro executor, motor, experiment configuration, experiment status, scan plots, Qt Spock, ...



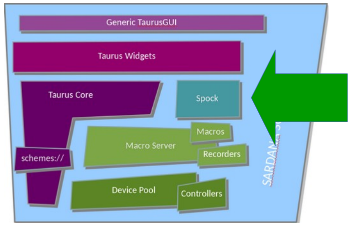
Taurus Core



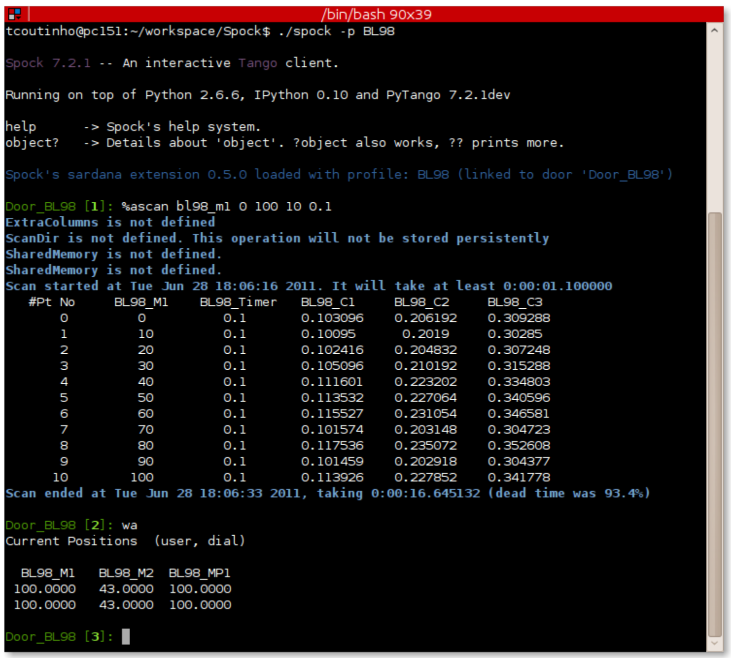
- “Container of unique models”.
- Schemes provide access to different type of data sources.
- Polling and event mechanism.

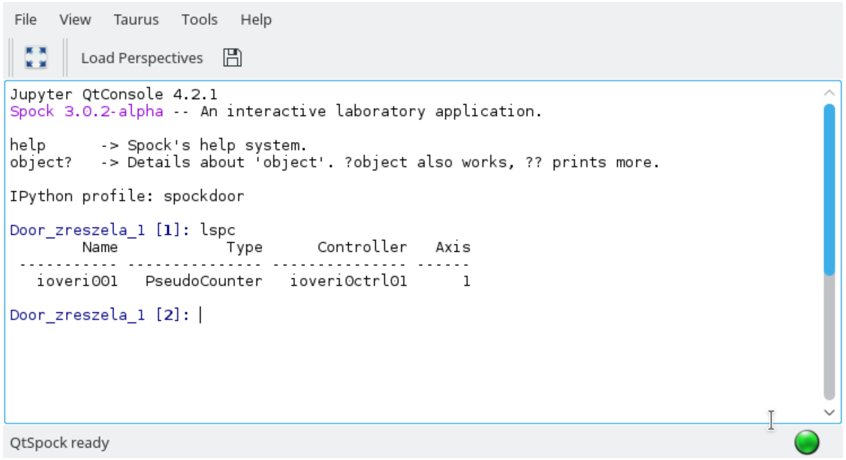


Spock

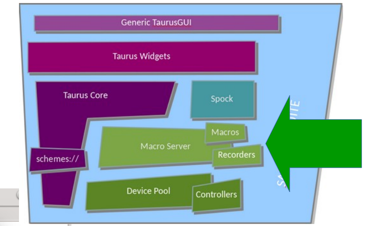


- **Spock** – **IPython** based Sardana CLI (also available as Qt widget).
- Provides total control over the system: executes procedures, interacts with the elements, …
- Spock syntax mimics **SPEC** commands.



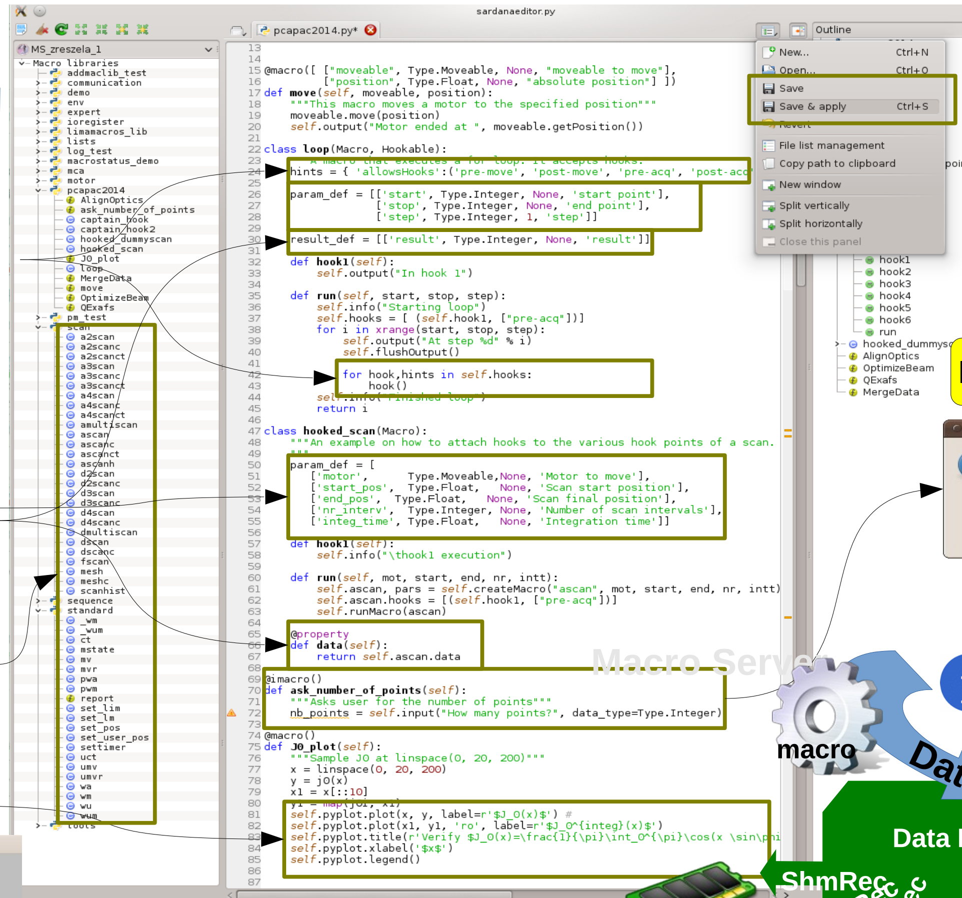


MacroServer



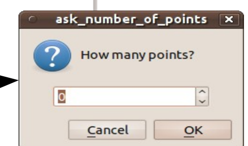
(General) Hooks

```
Door_1 [8]: loop 0 10 3
Starting loop
At step 0
running hook with hints=['pre-acq']
In hook 1
At step 3
running hook with hints=['pre-acq']
In hook 1
At step 6
running hook with hints=['pre-acq']
In hook 1
At step 9
running hook with hints=['pre-acq']
In hook 1
Finished loop
```



Adding, editing macros at runtime

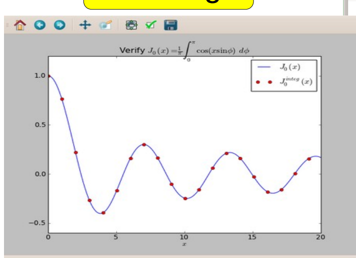
Interactive macros

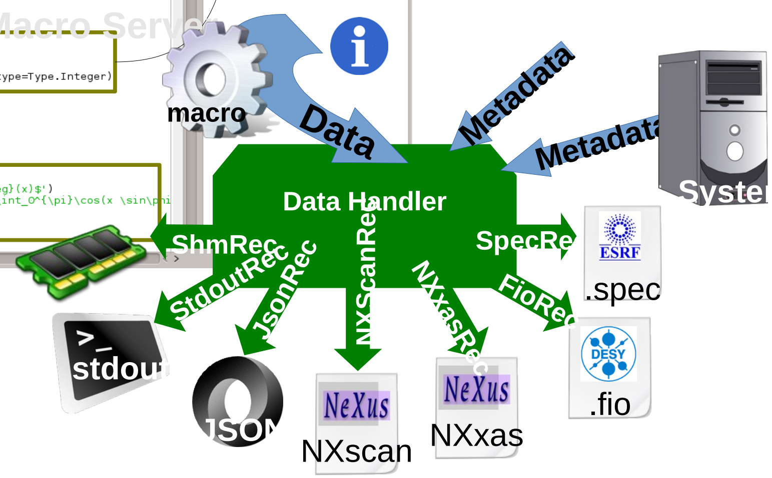


Input parameters & results & data

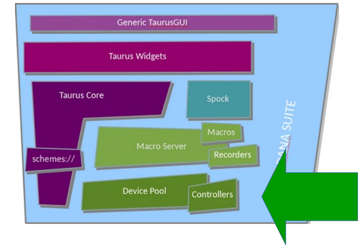
SPEC like commands

Plotting

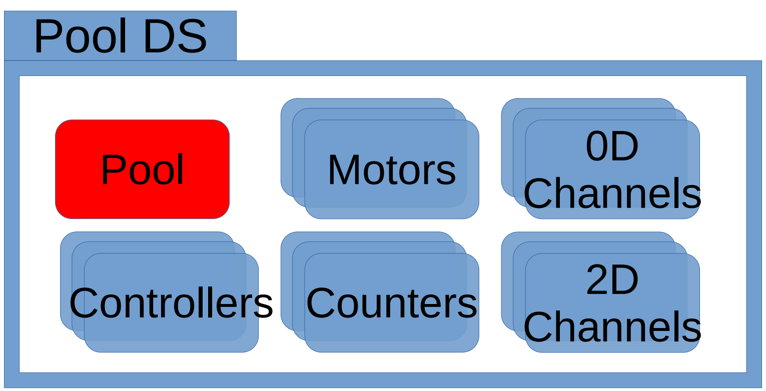




Device Pool



- All the equipments are interfaced via Device Pool and its **plug-in** controller classes (**Python**)
- Generic elements' interfaces allow building high level layers on top of them e.g. MeasurementGroup, pseudo elements, generic widgets, etc.



Pool Device Server and its elements

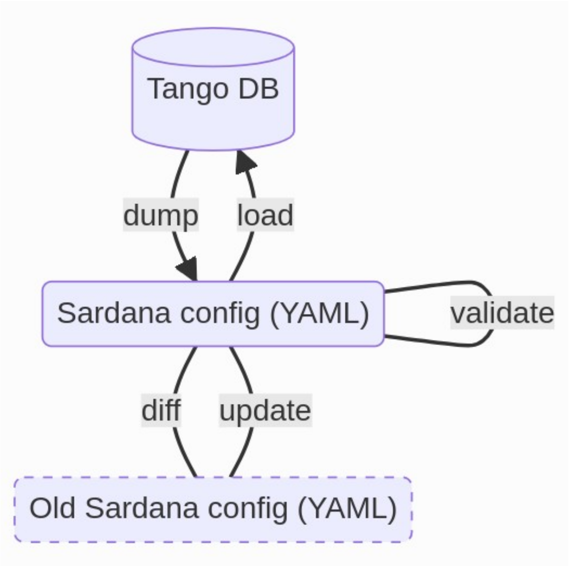
Element Type	*Example of application*
Motor	stepper, servo or piezo actuator
PseudoMotor	energy, HKL of a diffractometer, slit's gap or offset
CounterTimer	event counter, position measurement
PseudoCounter	vertical beam position in the X-ray beam position monitor (XBPM)
0DExpChannel	analog to digital converter (ADC), low current electrometer
1DExpChannel	position sensitive detector (PSD), multichannel analyzer (MCA)
2DExpChannel	CCD camera, 2D X-ray detector
TriggerGate	timing cards, pulse train generators

Configuration

- Sardana has multiple configuration points e.g. Tango DB, environment DB, etc.
- The idea is to unify all the configuration into one place.
- Format based on YAML.
- CLI scripts for configuration operations: sardana config --help


```yaml
tango_host: tangodb:10000

pools:
  demo1:
    instruments:
      /slit:
        class: NXcollimator
      /mirror:
        class: NXmirror
      /monitor:
        class: NXmonitor

    controllers:
      motctrl01:
        type: Motor
        python_module: DummyMotorController.py
        python_class: DummyMotorController
        elements:
          mot01:
            axis: 1
            instrument: /slit
          mot02:
            axis: 2
            instrument: /slit
          mot03:
            axis: 3
            instrument: /mirror
          mot04:
            axis: 4
            instrument: /mirror
```


Example of the Sardana configuration file



Sardana configuration tools

Introduction to Sardana

Continuous Scan Approach

Continuous Scan Details

Status & Roadmap

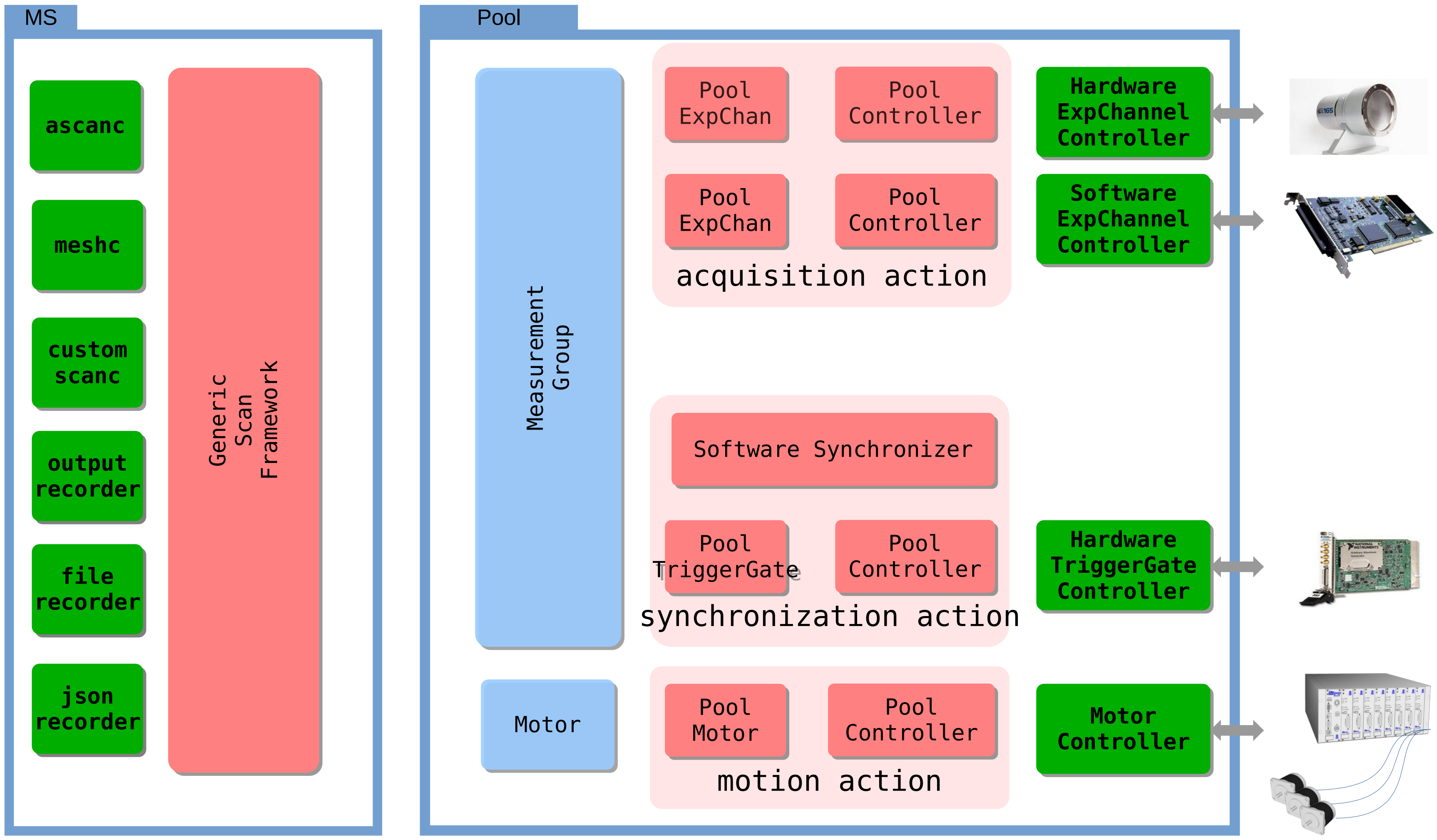
Objectives

- Provide a generic framework for continuous scans that would reduce the efforts of development of a particular scan in a laboratory.
- Ideally the only customizations needed:
  - hardware access plug-ins (controllers) development
  - configuration e.g. elements configuration, create and configure a measurement group, environment variables

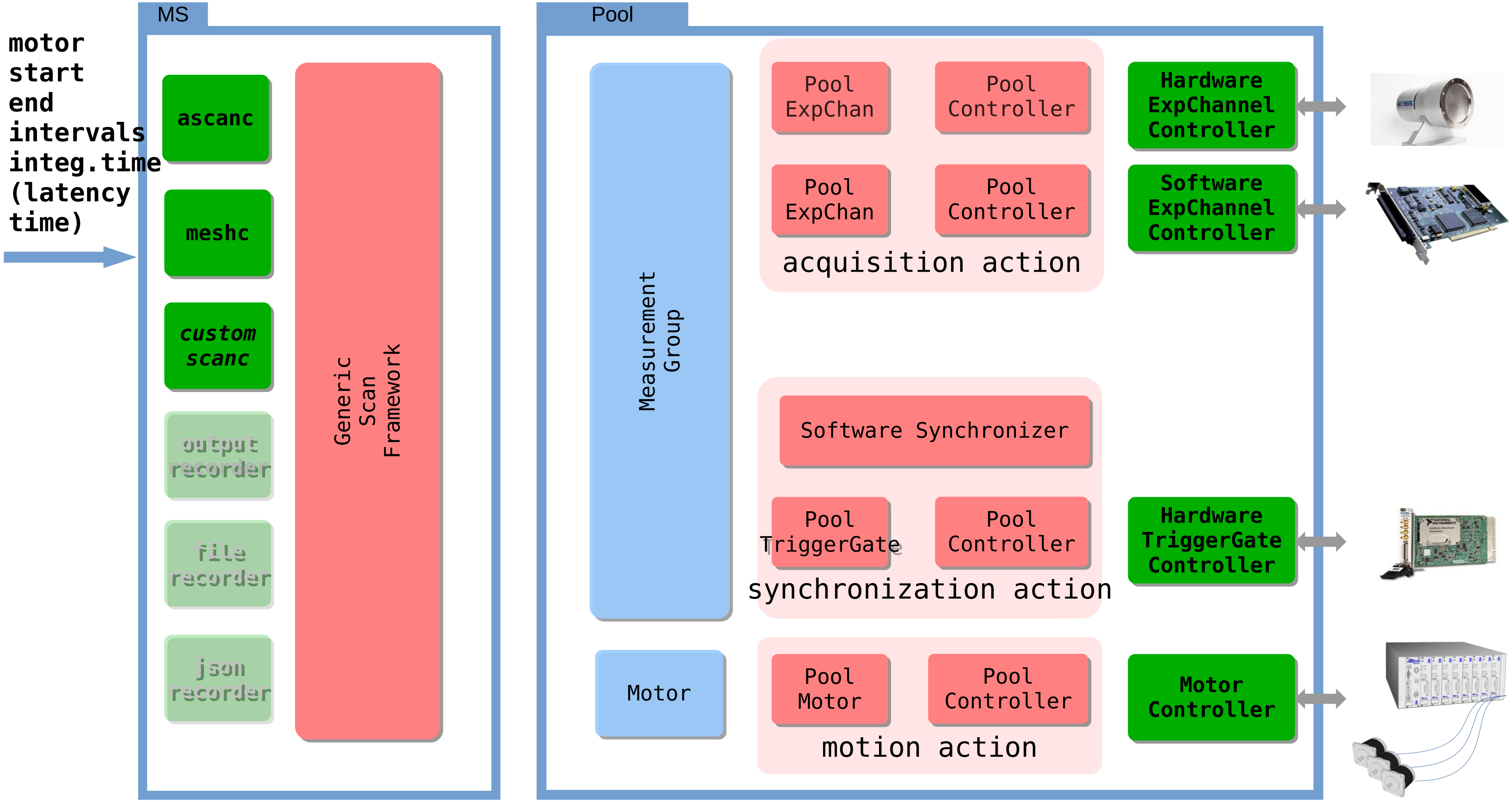
Important assumptions

- All the elements, but slave motors, must be defined in the same Pool.
- Maintain a unique synchronization description hence integration time for all the involved controllers per scan point.
- Experimental channel configurations e.g. repetitions or synchronization are set on the controller level.
- Support only linear trajectories – constant velocity of physical motors over the whole scan range.

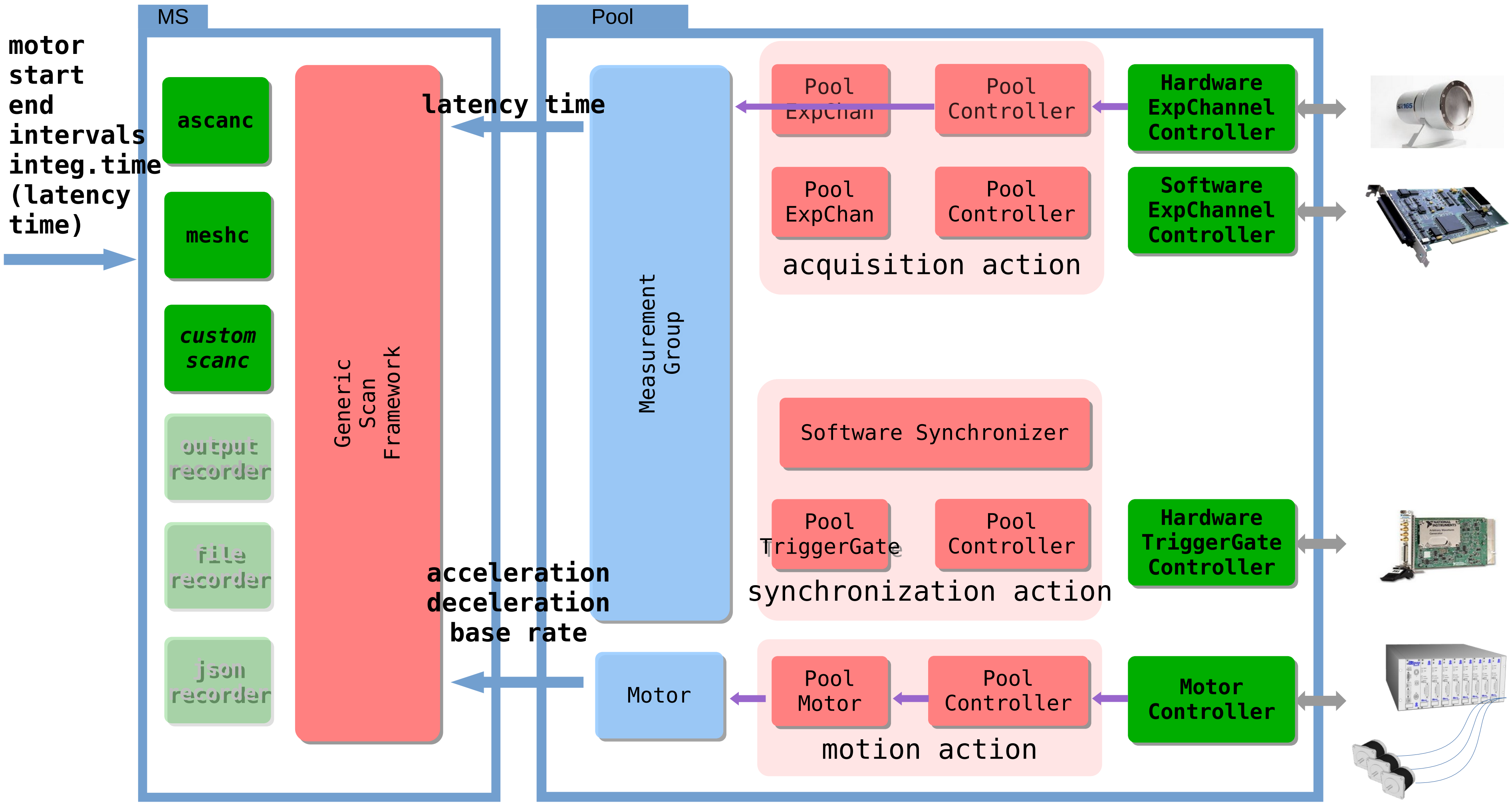
Involved elements



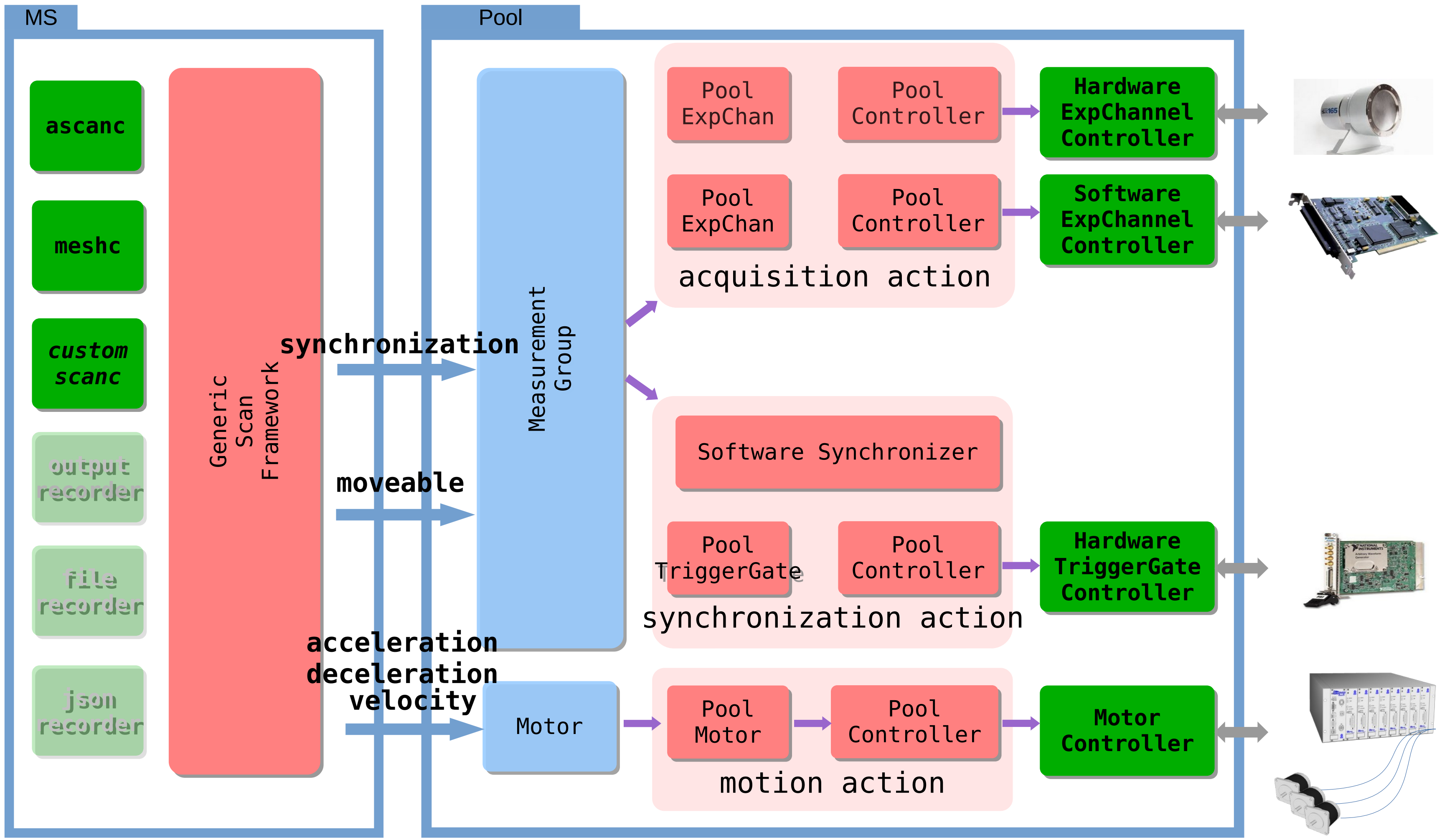
Scan configuration



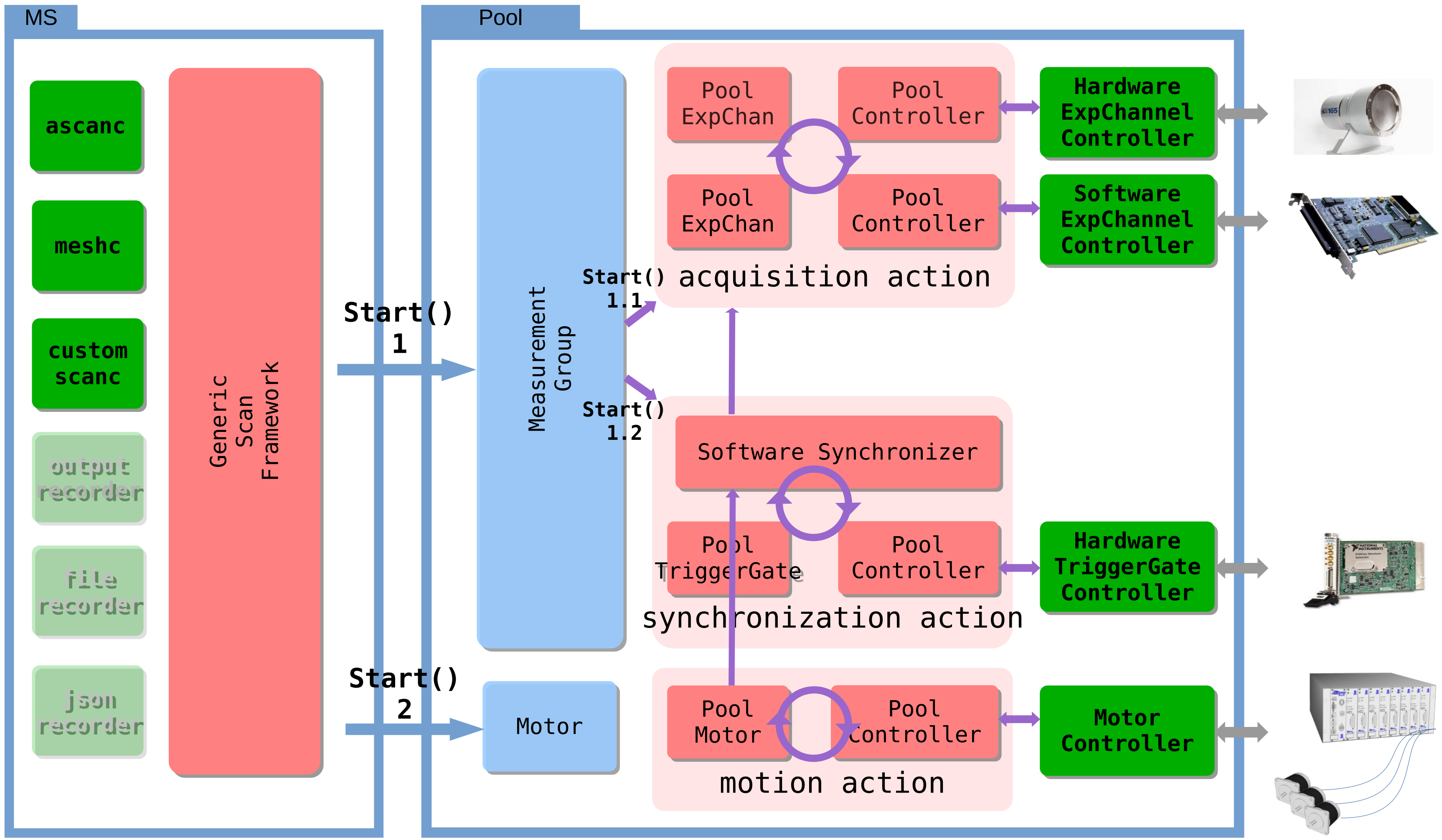
Scan configuration



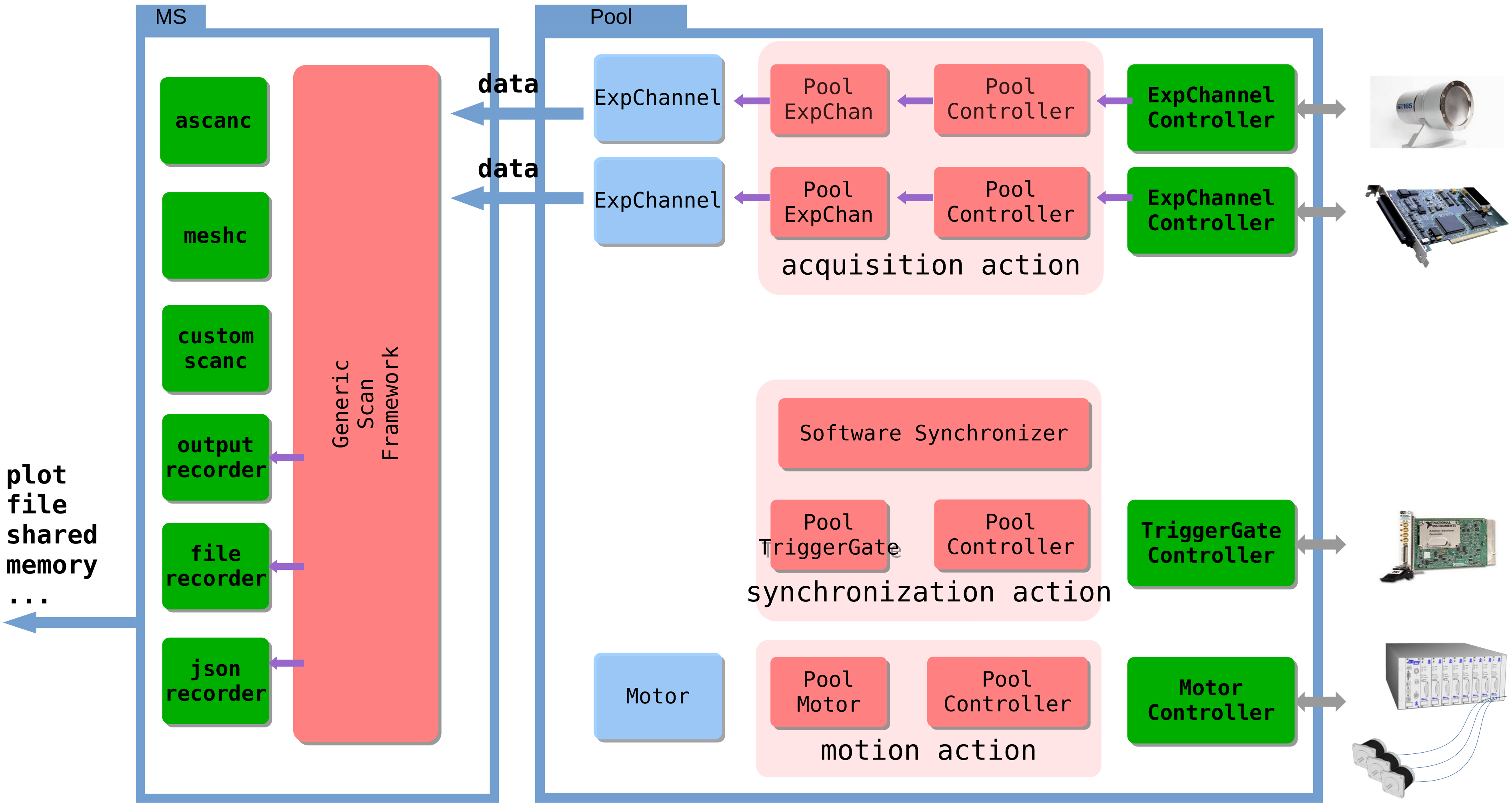
Scan configuration



Synchronization and acquisition



Data readout, merging & storage



Objectives & Assumptions

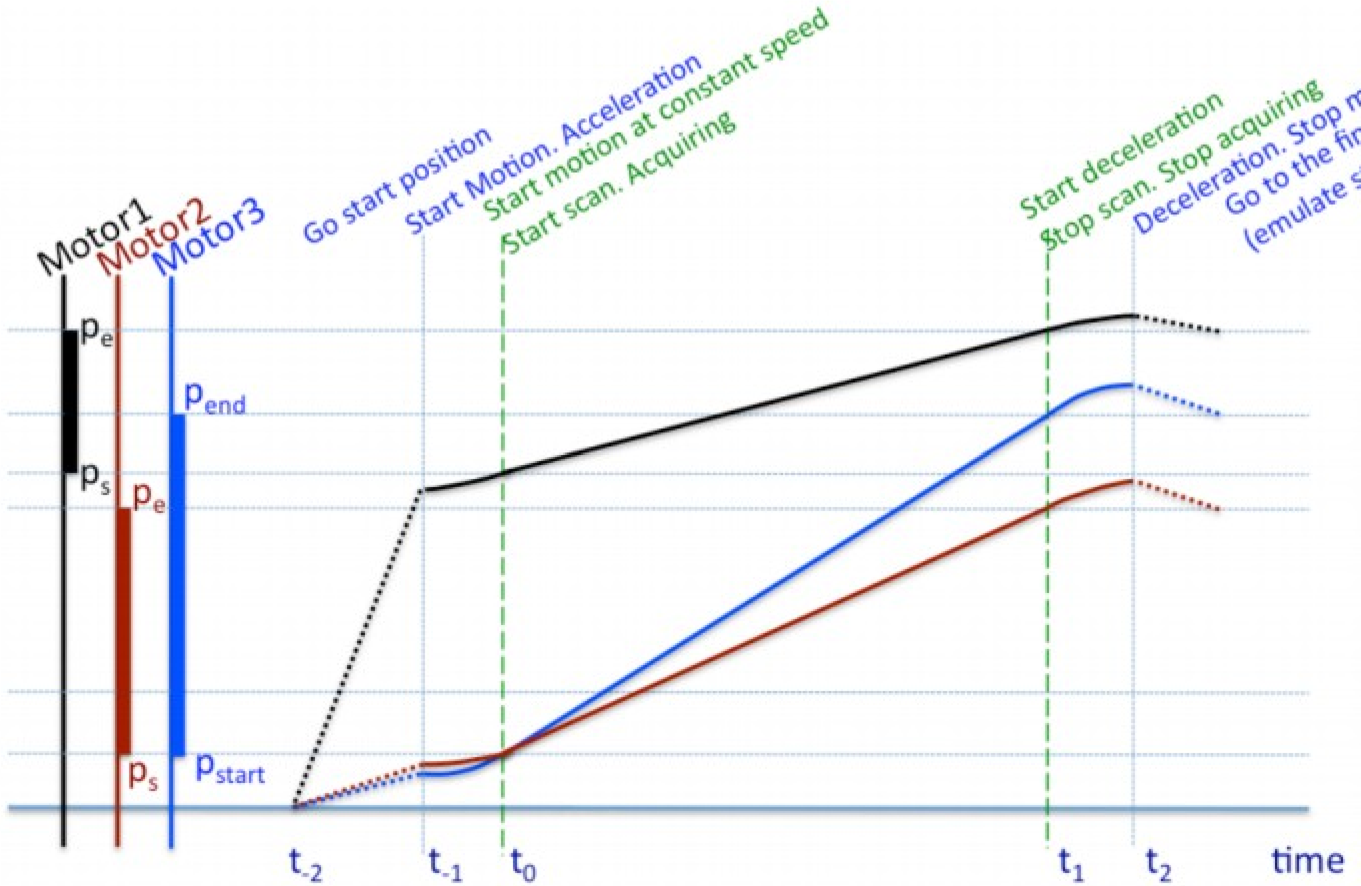
Continuous Scan Approach

Continuous Scan Details

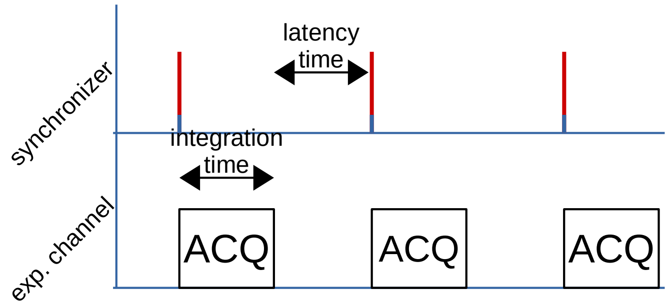
Status & Roadmap?

Motion

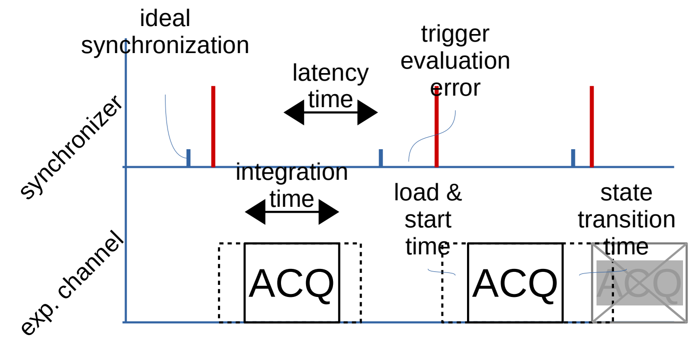
- Physical motors maintain constant velocities while scanning – **no trajectory control** (for the moment...).



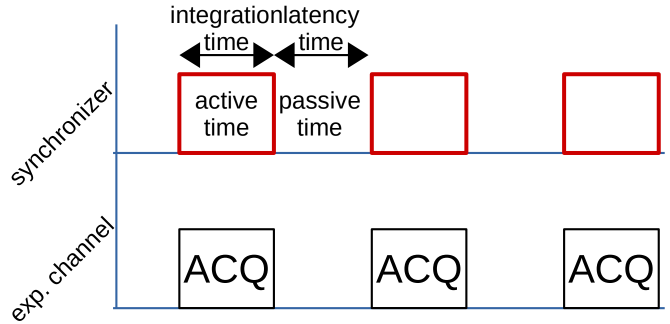
Synchronization Modes



hardware (external) trigger



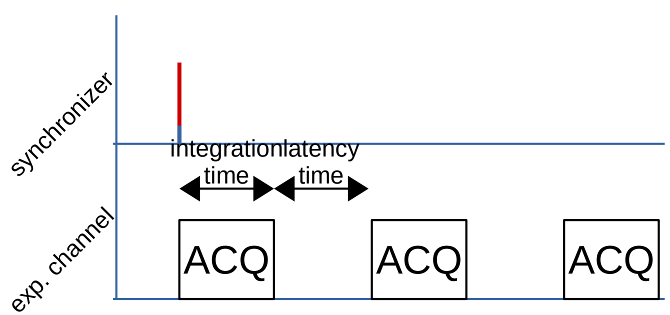
software (internal) trigger



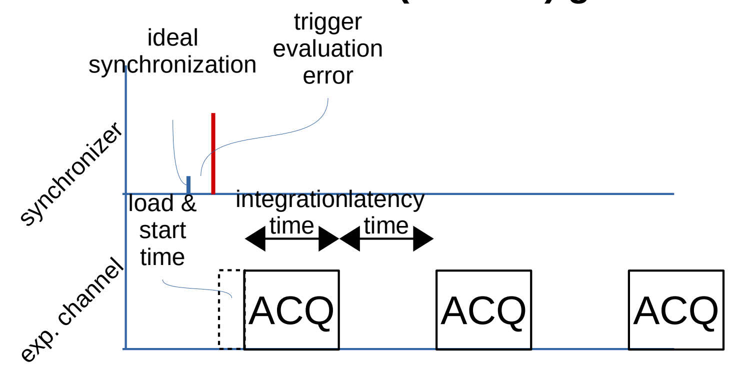
hardware (external) gate

```
raise NotImplementedError()
```

software (internal) gate



hardware (external) start

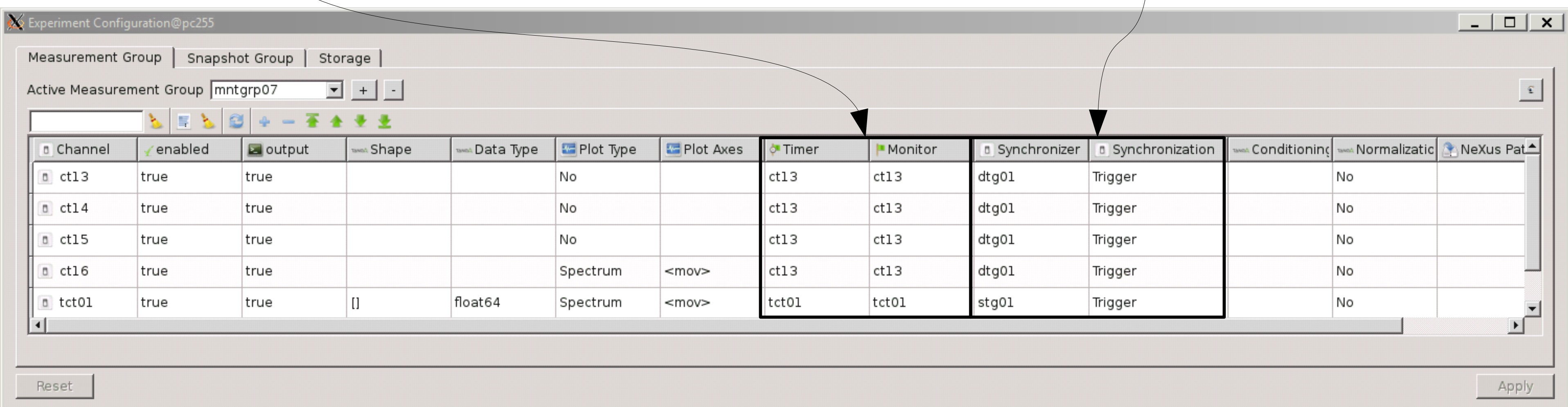


software (internal) start

Measurement Group

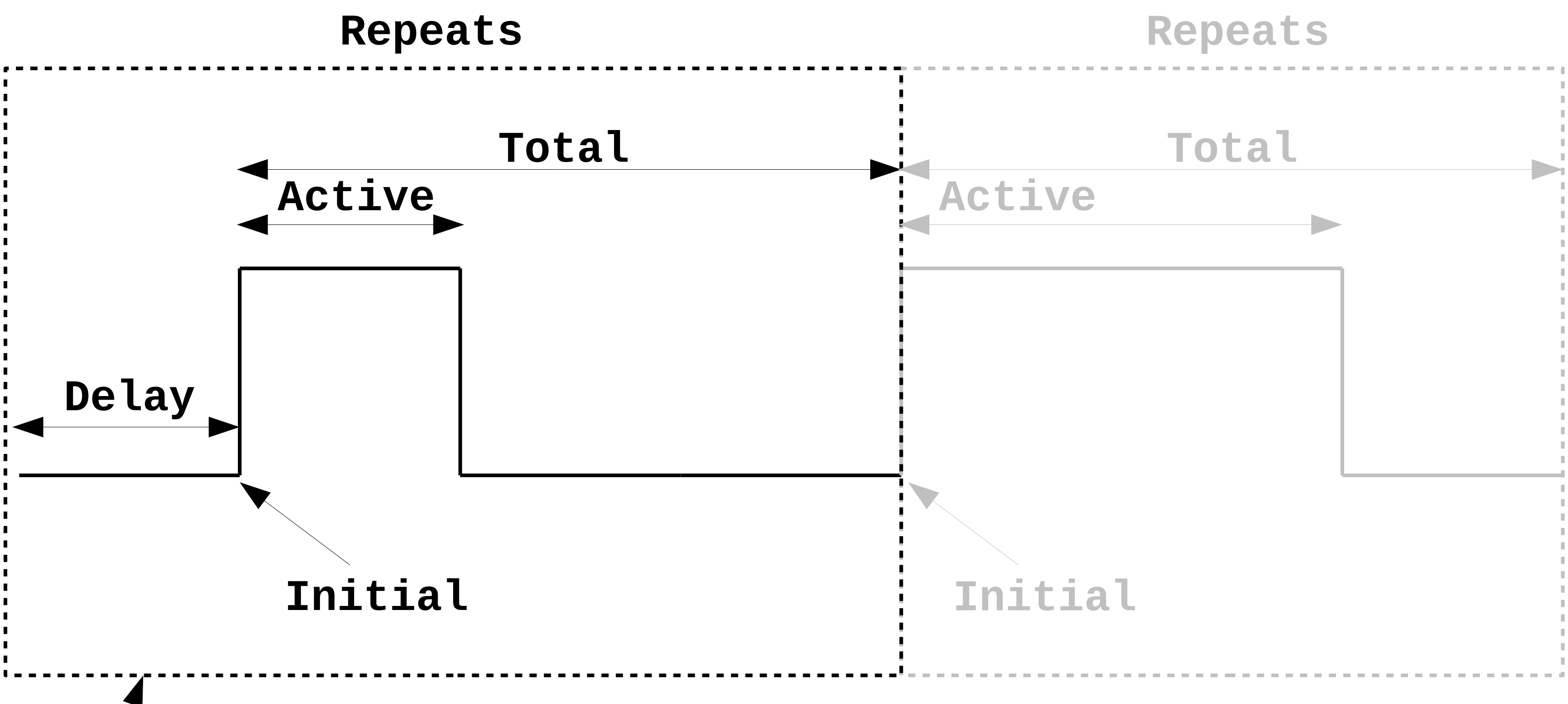
Timer and **Monitor** are maintained

Synchronizer and **Synchronization** (on the controller level)



Channel	enabled	output	Shape	Data Type	Plot Type	Plot Axes	Timer	Monitor	Synchronizer	Synchronization	Conditioning	Normalizatio	NeXus Pat
ct13	true	true			No		ct13	ct13	dtg01	Trigger		No	
ct14	true	true			No		ct13	ct13	dtg01	Trigger		No	
ct15	true	true			No		ct13	ct13	dtg01	Trigger		No	
ct16	true	true			Spectrum	<mov>	ct13	ct13	dtg01	Trigger		No	
tct01	true	true	[]	float64	Spectrum	<mov>	tct01	tct01	stg01	Trigger		No	

Synchronization description



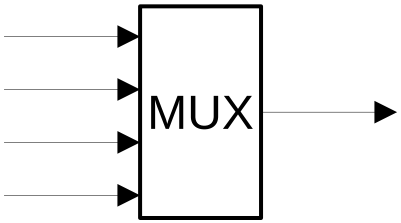
Group

```
[{Delay:   {Time: 0.3, Position: 400},
  Initial: {Time: None, Position: 0},
  Active:  {Time: 0.1, Position: 10},
  Total:   {Time: 0.15, Position: 15},
  Repeats: 1000},
 ...]
```

Position domain values in *dial position*

Trigger Gate Controller

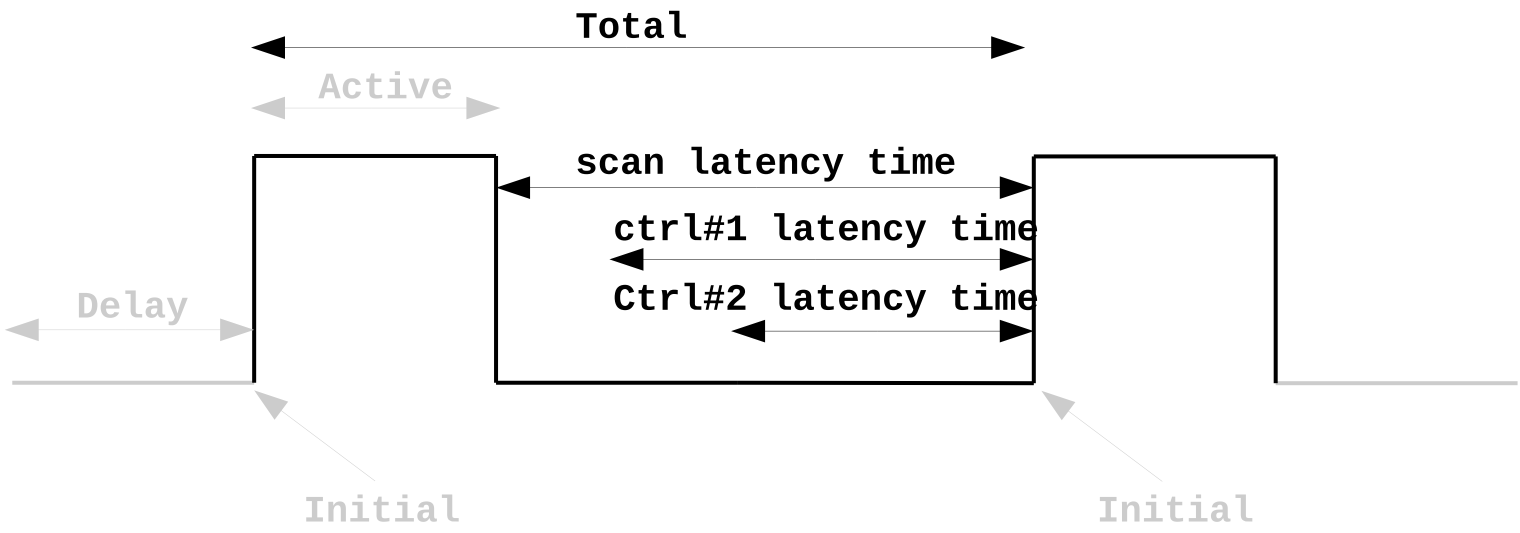
In case of supporting the multiplexor mode the **active_input** axis parameter selects the input.



`SynchOne()` sets synchronization description to the hardware

```
class MyTriggerGateController(TriggerGateController):

    def __init__(self, *args, **kwargs):
        TriggerGateController.__init__(self, *args, **kwargs)

    def SetAxisPar(self, axis, par, value):
        # set axis parameter: active_input

    def SynchOne(self, axis, description):
        repeats = 0
        for group in description:
            repeats += group[SynchParam.Repeats]
        # retrieve and set the rest of the parameters...

    def StartOne(self, axis):
        # start your channel

    def StateOne(self, axis):
        # read state

    def AbortOne(self, axis):
        # abort your channel
```

Latency time



- Latency time ensures that the acquisition goes smoothly.
- Affects: total interval (time), motors velocities.
- Controller latency time represents the re-arming time.
- Scan latency time is useful when software synchronization is involved – it helps to avoid skipped acquisitions.
- “passive time” = max(ctrl#1, ctrl#2 & scan latency times)

Exp. Channel Controller

Set controller parameter synchronization with one of **AcqSynch**:

- `SoftwareTrigger`
- `HardwareTrigger`
- `HardwareGate`
- `SoftwareStart`
- `HardwareStart`

`LoadOne()` accepts number of **repetitions** and **latency_time**.

Conditional programming.

`ReadOne()` returns either:

- single value
- list of values

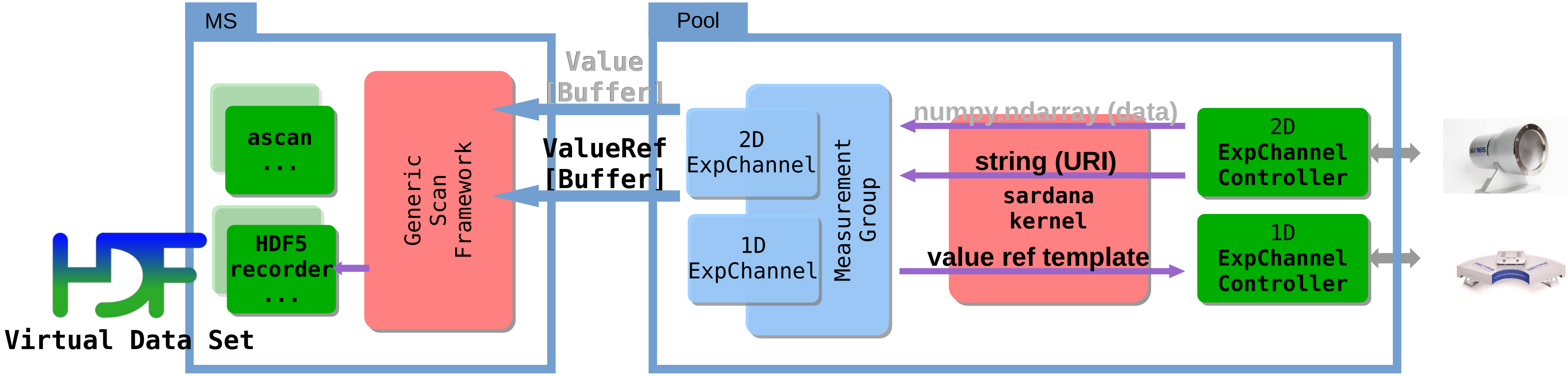
```python
class MyCounterTimerController(CounterTimerController):

    def __init__(self, *args, **kwargs):
        CounterTimerController.__init__(self, *args, **kwargs)

    def SetCtrlPar(self, par, value):
        # set controller parameter
        if par == 'synchronization':
            # set type of synchronization

    def GetCtrlPar(self, par, value):
        # get controller parameters
        if par == 'latency_time':
            # return latency time (re-arming time)

    def LoadOne(self, axis, integ_time, repetitions,
                latency_time):
        # load integration time and repetitions

    def StartOne(self, axis, value=None):
        # start your channel

    def ReadOne(self, axis):
        # read acquired data
        # if AcqSynch.HardwareTrigger return a list of values
        # if AcqSynch.SoftwareTrigger return a value

    def StateOne(self, axis):
        # read state

    def AbortOne(self, axis):
        # abort your channel
```

1D and 2D channels

- 1D and 2D channels can report references to the data (URI). Optional and configurable.
- When references point to HDF5 files the creates VDS.

```
class My2DController(TwoDController,
                     Referable):

    [...]

    # return one value or chunk with n values
    def ReadOne(self, axis):
        return numpy.array([[1,2],[3,4]])

    # return one ref or chunk with n refs
    def RefOne(self, axis):
        return “h5file:///tmp/image.h5::data”

    [...]
```

*Readable and **Referable** interfaces of the exp. channel controller.*



Introduction to Sardana

Continuous Scan Approach

Continuous Scan Details

Status & Roadmap?

Community

- Sardana and Taurus started as internal ALBA projects.
- In 2013 their governance was opened to the community of users and developers and were adopted in several other institutes.
- Community activities: regular follow-up meetings, hackathons, documentation camps, workshops, joined developments and debugging sessions.
- Big thanks to all the contributors that were involved in these 10 years!

Status

- Sardana Generic Scan Framework comes with support for Continuous Scans.
- The design and implementation dates 2013-2015 with some later enhancements e.g. 1D&2D data references, start synchronization, multiplexor mode, etc.
- Its limitations can be workaround with hooks and custom plugins.
- Sardana Continuous Scans are used at ALBA and MAXIV. SOLARIS also plans to use them.
- New requirements motivate us to think about new solutions and enhancements.

Roadmap?

- Allow **different synchronization descriptions** e.g. involving shutter control, synchronizing with the accelerator events
- Implement **trajectory control** using pseudomotors. Currently, only possible with motors which actually involve multiple physical actuators.
- Support for **multiple capability controllers** i.e motion and trigger gate in the same controller class – use capability composition approach.
- Improve support **high speed scans** - improve data flow to avoid bottlenecks e.g. pseudo counters, data storage, etc.
- Other minor ideas...

Thank you for your attention!

Docs: *www.sardana-controls.org*

- User’s Guide:
  → Scans
  → Standard Macro Catalog → ascanct, ...
  → Environment Variable Catalog → ApplyInterpolation, ...
  → Sardana-Taurus Widgets → expconf, ...
- Developer’s Guide
  → Measurement Group overview
  → How to C/T controller
  → How to T/G controller