# New Scanning Engine for ESRF Beamlines

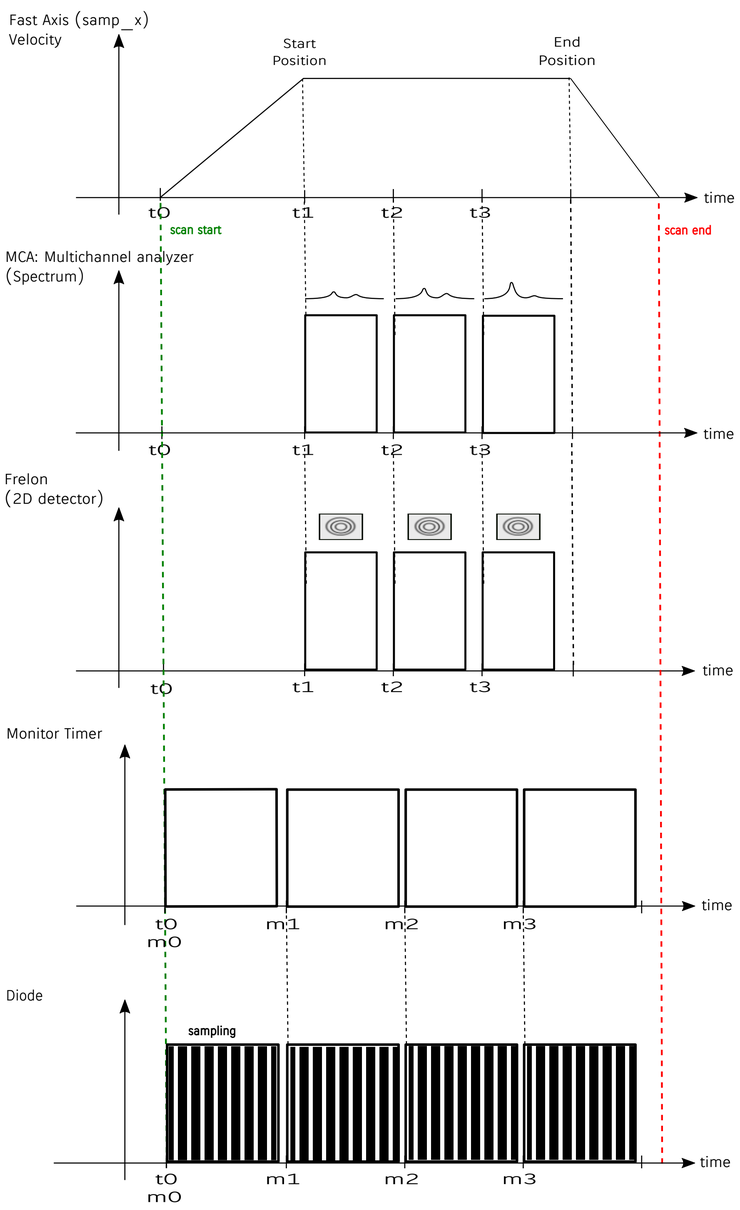
Sébastien Petitdemange, Antonia Beteva, Tiago Coutinho, Marie–Christine Dominguez, Matias Guijarro, Cyril Guilloud, Alejandro Homs, Jens Meyer, Vincent Michel, Emmanuel Papillon, Manuel Perez

**Grenoble, France**

BLISS is the new beamline experiments control system currently being developed at ESRF. At the heart of BLISS is an innovative scanning engine, designed to support a whole range of data acquisition procedures. The majority of scans performed at ESRF as of today can be written in a few lines of Python code thanks to well–defined concepts and a clean API.

## Example scan

A continuous scan of a fast axis, triggering data acquisition from a MCA and a 2D detector while a monitoring timer triggers diode readings




## Acquisition Chain

Tree structure to represent devices involved in a scan and how they are related.

**Master devices** encapsulate triggering controllers like motor controllers (position trigger), 2D detectors (readout trigger), or even software controllers **to define a sequence of triggers for the slave devices**

**Slave acquisition devices** encapsulate data acquisition controllers, and define **data channels** ; data is acquired following the sequence of triggers from the master

**Preset objects** manage the execution context of the acquisition chain during the scan, for example to open & close a shutter, or to configure a multiplexer

An acquisition chain can have multiple top masters, and within a branch masters and slaves can be nested up to any level

```python
def cscan(motor, start, stop, npoints, time):
    # create a new chain
    chain = AcquisitionChain(parallel_prepare=True)
    # create the monitor timer
    # npoints == 0 mean infinite
    monitor_timer = SoftwareTimerMaster(1., name="monitor_timer",
                                        npoints=0)
    # create acquisition device for the monitor diode
    diode_device = SamplingCounterAcquisitionDevice(diode1,
                                                    count_time=1.,
                                                    npoints=0)
    # Associate them in the chain
    chain.add(monitor_timer, diode_device)
    # Now the fast acquisition
    # create a motor master for a position trigger
    master = SoftwarePositionTriggerMaster(motor, start, stop, npoints,
                                           time=time)
    # The spectrum device MCA
    mca_acq = McaAcquisitionDevice(mca, npoints=npoints,
                                   trigger_mode=McaAcquisitionDevice.GATE,
                                   counters=list(mca.counters))
    chain.add(master, mca_acq)
    # The Image detector
    lima_master = LimaAcquisitionMaster(frelon,
         acq_nb_frames=npoints, acq_trigger_mode='EXTERNAL_GATE')
    lima_master.add_counter(frelon.image)
    chain.add(master, lima_master)
    # Finally build the scan and run it.
    scan = Scan(chain,name='cscan')
    scan.run()
    return scan
```

## BLISS Scan object

Runs the Acquisition Chain and configure the Data Streaming to perfom a scan

Scan loop is optimized to achieve the best synchronisation, taking account hardware and software triggering and data reading, in case a sync error is detected scan is interrupted

Timing statistics are recorded during the scan. This helps diagnosing problems with hardware devices and to debug new procedures

## Data Streaming

Data is **both** written to a HDF5 file and **published to a redis database** while the scan is running for on–line data visualisation and analysis

Data archiving and publishing paths are hold by a single configurable structure.

Any external program can connect to redis to access the data stream on–the–fly to perform data analysis or even online feedback (influencing the scan while it is running), if it has been written to support this feature

HDF5 file contains all data acquired and metadata (i.e: sample names, experiment name, all motor positions)

Voluminous data like 2D images are just referenced, whereas small data is directly stored in redis for some time (1 day by default). Using BLISS Python API to connect to the stream, data is delivered via the more efficient path, e.g directly from a Tango server memory, if it is still there, to save disk I/O