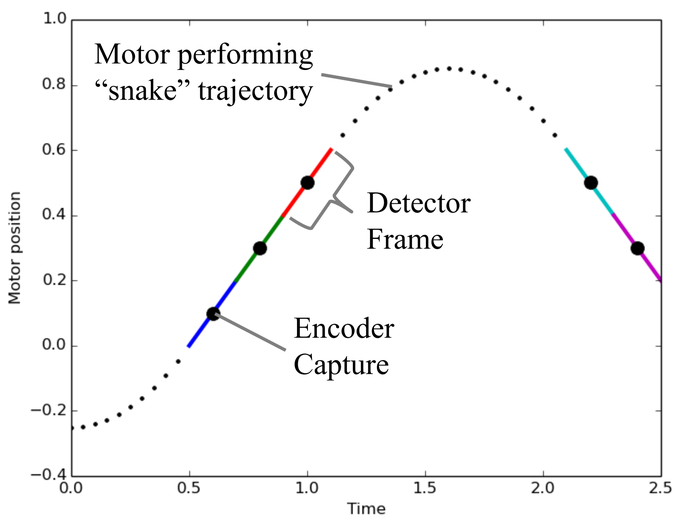
Malcolm: A Middlelayer Framework for Generic Continuous Scanning

T. Cobb, M. Basham, G. Knap, C. Mita, M. Taylor, G. D. Yendell, Diamond Light Source Ltd, Oxfordshire, UK,
A. Greer, Observatory Sciences, Cambridge, UK

Malcolm is a middlelayer framework that implements high level configure/run behaviour of control system components like those used in continuous scans. It was created as part of the Mapping project at Diamond Light Source to improve the performance of continuous scanning and make it easier to share code between beamlines. It takes the form of a Python framework which wraps up groups of EPICS PVs into modular "Blocks". A hierarchy of these can be created, with the Blocks at the top of the tree providing a higher level scanning interface to GDA, Diamond's Generic Data Acquisition software. The framework can be used as a library in continuous scanning scripts, or can act as a server via pluggable communications modules. It currently has server and client support for both pvData over pvAccess and JSON over websockets. When running as a webserver this allows a web GUI to be used to visualize the connections between these blocks (like the wiring of EPICS areaDetector plugins).

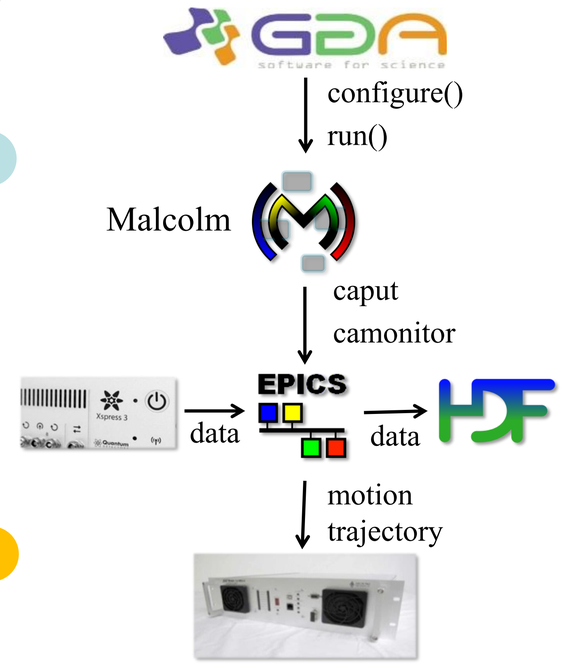
What is Continuous Scanning?

Continuous scanning is where motors are moved in a continuous trajectory while a detector takes a number of data frames synchronized with hardware trigger pulses.



This technique increases the efficiency of an experiment by reducing the number of times a motor has to decelerate, settle and accelerate, effectively decreasing the scan dead-time

What is Malcolm?

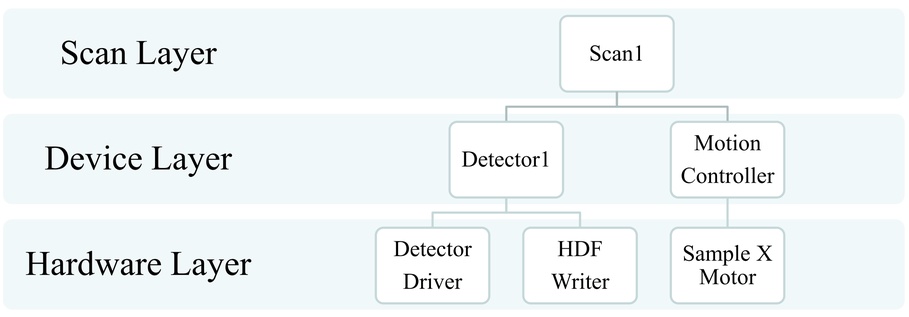


Malcolm provides an abstraction layer on top of EPICS that wraps up groups of PVs and presents a higher level scanning interface to GDA via pvAccess.

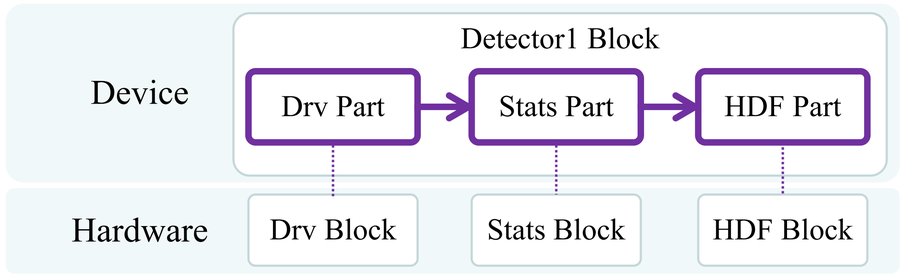
This means that it can take care of the variations in triggering schemes between different beamlines, and GDA only needs to pass high level scan parameters such as motion trajectory and exposure time down, rather than needing to know how all the underlying devices are wired up.

Blocks, Methods and Attributes

Malcolm defines layers of Blocks, each with a series of Methods and Attributes, much like instances of classes in an object oriented language.



- The Hardware Layer contains Blocks that are just a collection of Attributes. They correspond to EPICS devices like a single motor, or the areaDetector HDF writer plugin.
- The Device Layer contains Blocks that represent a whole Detector or Motor Controller. They have configure() and run() Methods. When these methods are called they co-ordinate their child Hardware Blocks to perform a scan according to the parameters they are passed.
- The Scan Layer at the top exposes a scanning interface to GDA. Blocks in this layer also have configure and run Methods that again co-ordinate their child Device Blocks to perform a scan.

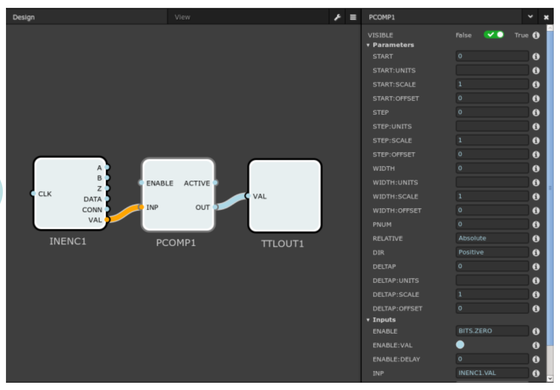


Each element in the areaDetector driver and plugin chain is defined by a Block in the Hardware Layer that defines the PVs it exports. The Device Block is then formed by composition from a single Controller and one Part for each child Block which contains the logic that shows how to use that Hardware Block within the current scan. This allows the external interface provided by PVs to be separated from small self-contained pieces of code that implement one particular type of logic.

Blocks are assembled from YAML and Parts can be activated and deactivated at run-time to change the components of a scan.

Communicating with Malcolm

Although Malcolm can be run standalone as a library, the most common use case is to add some communications modules to it to allow it to be communicated with from the outside.



Websocket communications expose the structure of Malcolm Blocks via a JSON protocol over websockets. There is a client MalcolmJS library that is used to create a web GUI to allow configuration/load/save of Blocks from a web browser, like in a PandABox.

pvAccess communications allow GDA to communicate with the top level scan Block, configuring it with a set of parameters then telling it to run. These are done using the pvaPy Python bindings to pvAccess in Malcolm, and the pvAccessJava bindings on the GDA side.

Serializing Malcolm Data

Block structures can be serialized to allow the web GUI to introspect Attributes and GDA to introspect configure arguments.

Attributes are conformant to an NTScalar because the value, alarm and timeStamp are present, but the metadata like descriptor, display and control have been moved to a Meta object so the same Meta objects can be used to specify arguments that should be passed to a Method.

Meta objects contain some of the meta information that would normally appear in the NTScalar, with some additions for specific Meta objects like the dtype (e.g. uint32) for the NumberMeta.

```
Block :=

malcolm:core/Block:1.0
    BlockMeta   meta
    Attribute   health
    {Attribute  <attribute-name>}0+
    {Method     <method-name>}0+
```

```
Attribute := Scalar | ScalarArray | …

Scalar :=

epics:nt/NTScalar:1.0
    scalar_t    value
    alarm_t     alarm       :opt
    time_t      timeStamp   :opt
    ScalarMeta  meta        :opt
```

```
ScalarMeta := NumberMeta | StringMeta …

NumberMeta :=

malcolm:core/NumberMeta:1.0
    string    dtype
    string    description
    string[]  tags       :opt
    bool      writeable  :opt
    string    label      :opt
    display_t display    :opt
    control_t control    :opt
```

For more information please contact
tom.cobb@diamond.ac.uk

diamond