EPICS Training

Scans

Tim Mooney
2/27/2015

Argonne National Laboratory
The synApps SSCAN module

- **Where is it?**

- **What’s in it?**
  - **Code**
    - the sscan record
    - the recDynLink library
    - the saveData data-storage client
    - the scanparm record
  - **EPICS databases**
    - scan databases
    - scanParms and alignParms databases
  - **MEDM displays**
    - scan*.adl
    - scan*_help.adl
Simple scans

- **A one-dimensional scan:**
  - Do NPTS times:
    - *Set conditions* e.g., move motors; wait for completion
    - *Trigger detectors* e.g., start scaler; wait for completion
    - *Acquire data* read detector signals; store in arrays
  - Write data to disk
...Simple scans

- Multidimensional scan:
  - Outer-loop scan’s *detector trigger* executes inner-loop scan.
  - *saveData* monitors a set of *sscan* records, determines scan dimension when scan starts, and writes data as it is acquired.
  - No limit to the number of scan dimensions.
Scan features

• 0-4 positioners, 0-4 detector triggers, 0-70 detector signals
  ▪ Positioner and readback values are of type `double`
  ▪ Detector values are of type `float`

• Acquisition from scalar and/or array PV's
  ▪ Array PV’s acquire .NPTS elements

• Number of data points limited only by IOC memory
  ▪ Standard max. is 2000 \((x_i,y_i)\) points per scan dimension
  ▪ Can increase to \(~\text{EPICS}_\text{CA}_\text{MAX}_\text{ARRAY}_\text{BYTES} / 8\)

• Detector/client wait, data-storage wait
  ▪ Can wait for multiple data-acquisition clients
  ▪ Only one data-storage client

• Pause/resume, abort
  ▪ Data from aborted scans are written to disk

• Double buffered: writes 1D acquired data after the scan is finished
  ▪ Can write during next 1D scan
...Scan features

- **saveData** writes XDR-format (".mda") files to disk.
  - Files can be read on any type of computer
- A **positioner can have private scan parameters (scanparm record).**
  - Load preset scan parameters with one mouse click
  - Useful for alignment
- **After-scan actions include move to peak, valley, +/-edge.**
  - Can, e.g., track a moving peak through a series of scans
- **scanparm record + after-scan action = automated 1-D alignment.**
Scan implementation

- **The sscan record is a channel-access client**
  - scanned PV’s can be hosted by any IOC
  - uses recDynLink library to manage connections with PV’s
  - uses `ca_put_callback()` to set conditions, trigger detectors, and await completion
  - uses `ca_get_callback()` before acquiring data

- **saveData is a channel-access client**
  - monitors sscan records and user-specified PVs
  - saveData can make the sscan record wait until data are written

- **Scan acquisition/storage can run on vxWorks, Linux, or Windows.**

- **The sscan record can be driven by any channel-access client.**
  - manual operation, via MEDM, is one option
  - often driven by spec and python code
  - can simplify user-written scan-control software
Before-scan / after-scan links

- Can write a constant value to any numeric or menu PV before the scan starts and/or after the scan ends.
- Can wait or not wait for completion of processing started by the write.
- If this scan record is part of a multidimensional scan, links function on each iteration.
- Outer-loop scan record can write to these links, and to the values they write.
- These links can write to their own scan record’s START, END, etc. fields, but not to its link fields.
MEDM user interface
Positioner options

- **SCAN MODE** (.PnSM - per positioner)
  - Determines how and to where positioner moves

- **Absolute/Relative** (.PnAR - per positioner)
  - Determines how positioner locations are written

- **Positioner delay** (.PDLY - affects all positioners)
  - Delay while positioners are settling, after completing their moves

- **After-scan motion** (.PASM - affects all positioners)
  - Determines what, if anything, is done with positioners when scan is finished
...Positioner options

- **SCAN MODE (\textit{.PnSM} - per positioner)**
  - **LINEAR** – Evenly spaced positions are calculated algorithmically
    - You specify positioner locations by setting any three of
      
      \[
      \begin{array}{cccccc}
        \text{START} & \text{CENTER} & \text{END} & \text{WIDTH} & \text{STEP SIZE} & \# \text{ POINTS} \\
      \end{array}
      \]
    
    - The sscan record reconciles unset parameters

  - **TABLE** – Positioner locations are contained in the \textit{.PnPA} array
    - The array must contain at least \textit{.NPTS} values
    - You must arrange for the array to contain the desired positions before starting the scan.
    - The \textit{.PnPA} array is never overwritten by the sscan record
...Positioner options

• ...SCAN MODE (.PnSM - per positioner)
  ▪ FLY – data will be acquired *while* positioner moves
    - You specify positions at which data are acquired by setting \textit{START}, \textit{END}, positioner speed, and detector acquisition time.
    - The following algorithm is executed:
      • Positioner sent to \textit{START}; reports completion
      • Detector triggered; reports completion
      • First data point acquired
      • Positioner sent to \textit{END}
      • \textit{NPTS}-1 iterations of
        • Detector triggered; reports completion
        • Data point acquired
    - The timing of data points is controlled by the detector’s acquisition time.
    - Fly-mode positioners do not report completion. (The positioner may still be moving after the scan ends.)
    - Note: timing of readback from a fly-mode positioner is inexact
Don’t be limited by existing positioner modes

- A positioner is *anything* you can write to
- Can specify positions algorithmically, using calcout or transform
  - E.g., sample-wheel
- Can write to positioner through interpolation table
  - Use a spare positioner readback to get actual positions into the data file
...Positioner options

- **Absolute/Relative (.PnAR - per positioner)**
  - If \texttt{.PnAR} == “ABSOLUTE” (0), positions are sent exactly as given.
  - If \texttt{.PnAR} == “RELATIVE” (1), positions are added to pre-scan position before being sent to positioner.

- **Settling time (.PDLY - affects all positioners)**
  - If any positioner PV is specified, then after all positioners report completion, the sscan record waits for \texttt{.PDLY} seconds before moving to next phase of sscan.
    - Useful for positioners that “ring” after move is completed
    - Useful work-around for positioners that cannot report completion
  - If no positioners, then settling time is ignored.
  - Settling time is adjusted to nearest multiple of system-clock period (typically 1/60Hz).
...Positioner options

- **After-scan motion (.PASM - affects all positioners)**
  - STAY – positioners are simply left where they ended up
  - START POS – positioners are sent to their START positions
  - PRIOR POS – positioners are sent to their pre-scan positions

- PEAK POS – data from the reference detector (number given by the .REFD field, in range [1..70]) is examined. If a peak is found, positioners are sent to where it was acquired.
- VALLEY POS – similar, but valley instead of peak
- +EDGE POS – peak of derivative of reference data
- -EDGE POS – valley of derivative of reference data
- CNTR OF MASS – center of mass: \( \frac{\sum x_i y_i \Delta x_i}{\sum y_i \Delta x_i} \). If more than one positioner is active, data from the lowest numbered active positioner will be used in the calculation. The result will, nevertheless, be applied to all active positioners.
Detector triggers

- 0-4 detector triggers (.TnPV), intended to start data-acquisition
- Similar to positioners, but value sent (.TnCD) is constant
- Triggers execute after all positioners have completed, and after any positioner settling time has elapsed.
- Detector settling time begins after all detector triggers have reported completion.
- If no triggers, then settling time is ignored.
Detectors

- PV’s to be acquired during scan
- 0-70 detectors (.D01PV - .D70PV)

Detector options

- Acquisition type (.ACQT)
  - SCALAR
    - scalar PV’s acquired at each positioner location
    - Array PV’s (.NPTS elements) acquired at end of scan
  - 1D ARRAY
    - use this mode only if ALL detectors are array valued
    - Positioners are only sent to their START positions.
    - In the future, array-valued positioners may be supported.

- Acquisition mode (.ACQM)
  - NORMAL – store values as acquired
  - ACCUMULATE – add detector values, starting with next scan
  - ADD TO PREV – same, but starting with previous scan
Client wait

- After all detector triggers have reported completion, and before acquiring data, the sscan record checks for client waits.
- Clients can hold scan at this point by writing ‘1’ to .WAIT (this increments the wait-count field, .WCNT)
- Several clients can use this field
- When all clients have written ‘0’ to .WAIT, scan acquires data.
- If clients are too slow to write to .WAIT, scan can set .WCNT for them, to the value .AWCT.
- Scan will pause until .AWCT clients have written ‘0’ to .WAIT.
- ‘Client’ includes user, via MEDM
Array trigger/wait

- After all data points have been acquired, scan can trigger software that prepares array PVs for acquisition (e.g., read from hardware).
- When array trigger declares completion, array PVs are acquired.
Scan controls

- **SCAN**
  - Writing ‘1’ starts this sscan record
  - Writing ‘0’ stops this sscan record. (But with the supplied database, always use the ‘ABORT’ button to stop.)

- **GO/PAUSE**
  - Pause is immediate, Go occurs after delay

- **ABORT**
  - Writes ‘1’ to ‘xxx:allstop.VAL’, which should stop motors
  - Sends “stop” message to all sscan records in the supplied database
    - First ‘Abort’ attempt ends scan after outstanding completion callbacks have come in, and data-storage client has released the previous scan’s data arrays.
    - Second ‘Abort’ attempt waits only for data-storage client.
    - Third successive ‘Abort’ attempt kills scan with no regard for consequences.
Scan user documentation

**MEDM displays**

This display controls a one-dimensional scan or one dimension of a multi-dimensional scan in multi-dimensional scans. In a one-dimensional scan, the following sequence of actions occurs:

1. Write to before-scan PV, and (optionally) wait for completion.
2. Generally, NPTS iterations of:
   - Write to positioners.
   - Wait for positioners to declare themselves 'done'.
   - Wait for positioner-setting time.
   - Write to detector triggers.
   - Wait for triggers to declare 'done'.
   - Wait until client wait count is zero.
   - Wait for detector-setting time.
   - Read positioner-readback and detectors, if acquisition mode is "ACQUIRE" or "READ-TO-FIRE". New data is added to last scan's data.
3. Write to array-read trigger, wait for completion, and read any array-valued detector signals.
4. Wait for data-storage client to finish saving last scan's data, and that executes after all scalar acquisition is done. Intended to any array hardware to pass data.

 Mostly status and identifying info. The number of data points is also specified here.

State of data according to scan record "WRITED" means it's been sent to data files. Info about scavenge — the program that monitors scans and writes data files.

Command that executes before any positioners move. Usually, a write causes some processing to occur, and you check whether the scan should wait for that processing to complete before proceeding to the next step.

Positioner set conditions under which data will be acquired. For example, you can move setoffs, set amplifier gains, etc. After all positioner commands have been sent, the scan waits for positioners to declare themselves done, and then waits for any programmed settling time before triggering detectors.

Detector triggers are like positioners, but they are intended to start data acquisition and do not declare themselves 'done'. A write to trigger triggers a detector and or data-acquisition client. If the trigger is used to trigger initiation of data-acquisition client, then the detector setting time "T".

Detector triggers are like positioners, but they are intended to start data acquisition and do not declare themselves 'done'. A write to trigger triggers a detector and or data-acquisition client. If the trigger is used to trigger initiation of data-acquisition client, then the detector setting time "T".

Detector triggers are signals read after all triggers and data-acquisition clients have completed. Any readback numeric PV can be named as a detector. If an array-valued PV is named, its array values will be acquired. (If all PV's are array-valued, you can set the acquisition type "DIMENSION" to "ARRAY").

Detector triggers are like positioners, but they are intended to start data acquisition and do not declare themselves 'done'. A write to trigger triggers a detector and or data-acquisition client. If the trigger is used to trigger initiation of data-acquisition client, then the detector setting time "T".

Detector triggers are like positioners, but they are intended to start data acquisition and do not declare themselves 'done'. A write to trigger triggers a detector and or data-acquisition client. If the trigger is used to trigger initiation of data-acquisition client, then the detector setting time "T".

Detector triggers are signals read after all triggers and data-acquisition clients have completed. Any readback numeric PV can be named as a detector. If an array-valued PV is named, its array values will be acquired. (If all PV's are array-valued, you can set the acquisition type "DIMENSION" to "ARRAY").

Detector triggers are like positioners, but they are intended to start data acquisition and do not declare themselves 'done'. A write to trigger triggers a detector and or data-acquisition client. If the trigger is used to trigger initiation of data-acquisition client, then the detector setting time "T".
One-click scans

- The scanparm record executes preprogrammed *linear scans*
  - Holds scan parameters for a positioner
  - Writes parameters to a particular `sscan` record
  - Optionally executes the `sscan` record
  - Useful for alignment
Data storage

- **saveData** monitors sscan records and writes their data to numbered files.
- **Handshake** permits pipelined operation.
- **saveData**’s boot-time init can specify list of PV’s to write with every scan’s data.
- **saveData** writes “MDA” files
  - MultiDimensional Archive
  - Binary, cross-platform (XDR) format
  - Format is optimized for run-time access.
  - Format permits file to be closed after each set of writes.
- **Automatic file numbering**
  - e.g., ‘xxx_0123.mda’, ‘xxx_0124.mda’
  - overlap is handled: ‘xxx_0123.mda_01’
Data storage

- **Location of data files**
  - ‘File system’ + ‘subdirectory’
  - vxWorks:
    - File system is NFS-mount point
    - ‘//<hostname>’ is required
  - Linux, etc.:
    - saveData doesn’t mount the file system (system administrator does this)
    - ‘//<hostname>’, if present, is ignored

- **Cannot write to ‘File system’ or ‘subdirectory’** while a scan is in progress. (See ‘LOCK’ PV.)
- Don’t delete or rename the directory saveData is writing to.
- Comment PV’s saved only if they are named in saveData.req
### saveData.req init file

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>[prefix]</strong></td>
<td>$(P)$</td>
</tr>
</tbody>
</table>
| **[status]** | $(P)$saveData_status ...
| **[scanRecord]** | $(P)$scanH $(P)$scan1 $(P)$scan2 $(P)$scan3 $(P)$scan4 |
| **[extraPV]** | #<PV name> <description>
$(P)$scaler1.TP "scaler preset (s)"
$(P)$scaler1.NM1 "scaler chan 1 desc" ...

- **List of sscan records to monitor**
- **List of PV’s to be saved with every scan** *(Normally, this is the only section you modify.)*
- **Description**
  - If not supplied, .DESC field is used
MDA file format

- **MDA FILE**
  - header
  - scans
  - extra

- **Header**
  - format version
  - scan number
  - scan dimension
  - (dim1, dim2, ...)
  - is regular scan
  - pointer to extra

- **Extra**
  - # extra PV’s
  - PV
  - PV
  - ...

- **PV**
  - name
  - description
  - type
  - count
  - units
  - value
...MDA file format

MDA FILE

header
scan
(scan)
extra

scan head
positioner 1
...detector 1
...trigger 1
...
data

Scan

scan rank
npts
actual pts
array of npts
detectors
triggers

data
# positioners
# detectors
# triggers
### MDA file format

**Scan**
- scan head
- positioner
- ...
- detector
- ...
- trigger
- ...
- data

**Positioner**
- number
- name
- description
- scan mode
- units
- readback name
- readback desc.
- readback units

**Detector**
- number
- name
- description
- units

**Trigger**
- number
- name
- command
...MDA file format

<table>
<thead>
<tr>
<th>MDA FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
</tr>
<tr>
<td>scan</td>
</tr>
<tr>
<td>extra</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan head</td>
</tr>
<tr>
<td>positioner 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>detector 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>trigger 1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>positioner 1 array</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>detector 1 array</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Other data-acquisition-related software

- Data-visualization tools for use with synApps
  - scanSee
  - mdaExplorer (synApps utils)
  - dview
  - sview
  - webics https://github.com/djvine/webics

- areaDetector
  - Allows any EPICS CA client to drive data acquisition.
  - Support ca_put_callback(), as needed by the sscan record.

- software to read, manipulate .mda files
  - mda.py
  - mdautils
Completion reporting

• Simple prescription for databases contained within a single IOC:
  ▪ Use only PP links and forward links in execution chain.

• Database operations spanning more than one IOC:
  ▪ Use records with put_callback links to span IOCs:
    - *calcout* with asynchronous device support
    - *sscan, swait* (i.e., a synApps “userCalc”)
    - *sseq* or *sCalcout* (with .WAIT* = “Wait”)

• Cases in which a CA client performs part of the operation:
  1) Database sets a **busy** record via PP or put_callback link.
  2) CA client clears the **busy** record when operation is done.

• Cases in which part of the operation is driven by a CP link:
  - Not different from above; a CP link is a CA client
...Completion reporting

- Use only PP links and forward links in execution chain.

![Diagram showing SCAN, PUT_CALLBACK LINK, and PP LINK with 'BUSY' and 'DONE' flags.](image)
...Completion reporting

- Same as before, but with an *asynchronous* record
Premature “DONE” report, because CA-link execution is not traced
...Completion reporting

- Premature-DONE problem fixed with a PUT_CALLBACK link

![Diagram showing PUT_CALLBACK LINK and PP LINK with "BUSY" and "DONE" flags]

Getting Started with EPICS IOCs: Scans
...Completion reporting

- Premature “DONE” because CA-client processing is not traced

```
PUT_CALLBACK LINK
PP LINK
CA PUT
CA MONITOR
```

![Diagram showing completion reporting process with flags indicating "BUSY" and "DONE" states.]

Getting Started with EPICS IOCs: Scans
...Completion reporting

- Premature “DONE” problem fixed with a ‘BUSY’ record

```
<table>
<thead>
<tr>
<th></th>
<th>PUT_CALLBACK LINK</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PP LINK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CA PUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CA MONITOR</td>
<td></td>
</tr>
</tbody>
</table>
```

```
CAN
  SCAN
    PUT_CALLBACK LINK
      PP LINK
        CA PUT
          CA MONITOR
            BUSY

CA CLIENT
```

Getting Started with EPICS IOCs: Scans
...Completion reporting