



Experimental Physics and Industrial Control System

EPICS

An Introduction

What is EPICS?

- EPICS is a software framework which allows us to design and implement a control system
 - Distributed open architecture, highly scalable
 - Control systems for small test stands
 - Control systems for very large facilities.
- It is *ideally* suited to the 'slow' controls (~ 50 ms+) for large, distributed control systems:
 - high reliability
 - high maintainability during long operational lifetimes (20 years+).
- The *EPICS collaboration* consists of the controls groups of many research organizations that use and develop the EPICS tool-kit.

History of EPICS

- Ronald Regan – USA President 1981-1989
 - The Strategic Defense Initiative or “Star Wars” (to shoot down nuclear missiles in space...)
- Lead, in 1988, to:
 - Development of a: 100 MeV, 100 mA pulsed-ion-beam accelerator (GTA) at Los Alamos.
 - Bob Dalesio and Jeff Hill worked on a new control system
- About this time, the Advanced Photon Source (APS) was being designed in Chicago
- Marty Kraimer joined Bob and Jeff and turned the Control System into EPICS

What is EPICS?

- EPICS is open-source software:
 - development and support depends on 'voluntary' effort
 - Occasionally financed by collaboration members
- EPICS provides a set of tools that reduces software application and maintenance costs by providing:
 - Configuration tools in place of programming
 - A large installed base of tested software
 - A modular design that supports incremental upgrades
 - Well defined interfaces for extensions at every level

The EPICS Collaboration

- Now well over 200 independent projects Worldwide
- Applications in:
 - Synchrotrons/beamlines (e.g. APS, Diamond, Australian Synchrotron)
 - Astronomy (e.g. Keck, Gemini, LIGO)
 - Neutron Sources (ESS, Oak Ridge, ISIS)
 - Fusion Energy (e.g. ITER)
 - High-energy Physics (IHEP China)

The EPICS Collaboration

- Problem reporting and resolution via e-mail exploders (e.g. 'tech-talk' list)
- Documentation available from download sites
<https://epics.anl.gov>
- Large collaboration meetings:
 - Training days
 - Status reports from various projects
 - Report new work that may be generally interesting
 - Plan future directions
 - Explore new requirements
 - Normally two per year: North America / Rest of the World
 - Next one: Pohang Accelerator Laboratory, Korea,
15th-18th April 2024

EPICS Software Distribution

The EPICS software distribution is in two parts:

EPICS **Base**. This is the main core of EPICS:

- The Build System
- Common and O/S interface libraries
- Defined, tested and released by Argonne National Lab
- Diamond currently uses: 3.14.12.7 (December 2017)
- Latest V3 stable release is: 3.15.9 (May 2020)
- EPICS 7 (V3 + V4): 7.0.8 (December 2023)

EPICS Software Distribution

EPICS ***Extensions***. Is a large and more loosely defined set of tools (mostly run on the host side):

- GUI's
- Archiving
- Monitoring tools

Extensions developed and distributed by individual organisations (GitHub)

Depending on local preferences, each EPICS site will have its own set of installed extensions.

No centrally organized distribution:

- there are diverse levels of support and compatibility.

EPICS Hardware Support

Hardware specific software is separately distributed:

- supports a wide range of devices
- different I/O buses i.e. VME, Profibus, Modbus, EtherCAT, Ethernet, etc
- <https://epics.anl.gov/modules/bus.php>

EPICS - A Distributed Architecture

- EPICS is a physically flat architecture of:
- Front-end controllers (IOC's "Input Output Controllers")
- Operator workstations (that run GUI's and tools)
- These communicate using TCP/IP and UDP protocols (Channel Access)
- EPICS software architecture is client/server based (no central nameservers, only need name of data point to find it)
- System scales simply through the addition of new IOC's and workstations
- If a single IOC becomes saturated, its functionality can be spread across multiple IOC's
- Network bandwidth is the primary limiting factor

EPICS Attributes

❖ Tool Based

- EPICS provides a number of tools both for creating and operating a control system
- Minimizes the need for custom coding
- Ensures uniform operator interfaces.

❖ Scalable

- IOC's from a few to > 500,000 "Process Variables" supported.

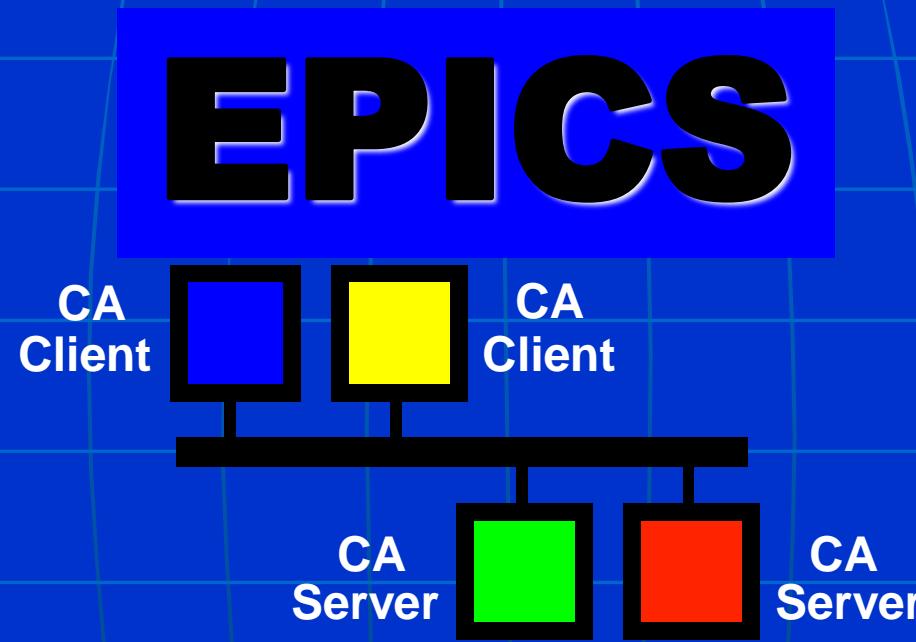
❖ Event Driven

- Only changes to a datum are sent from servers to clients
- Minimizes message latency and network loading

❖ Portability

- Runs on VxWorks, RTEMS, Linux, Windows PCs and macOS

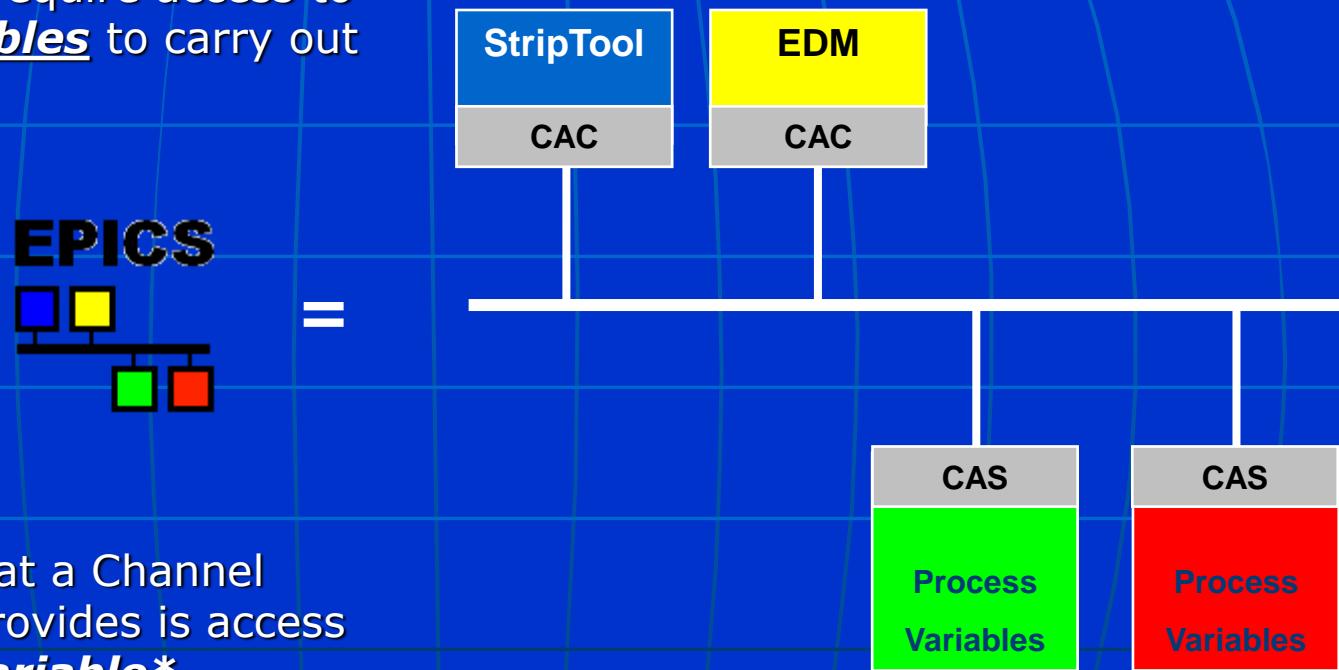
EPICS Logo: A Distributed Architecture



CA = Channel Access

EPICS on the Network

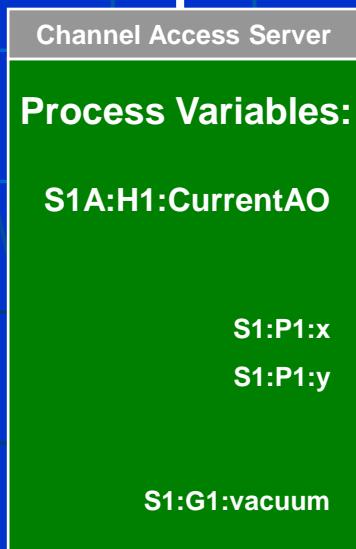
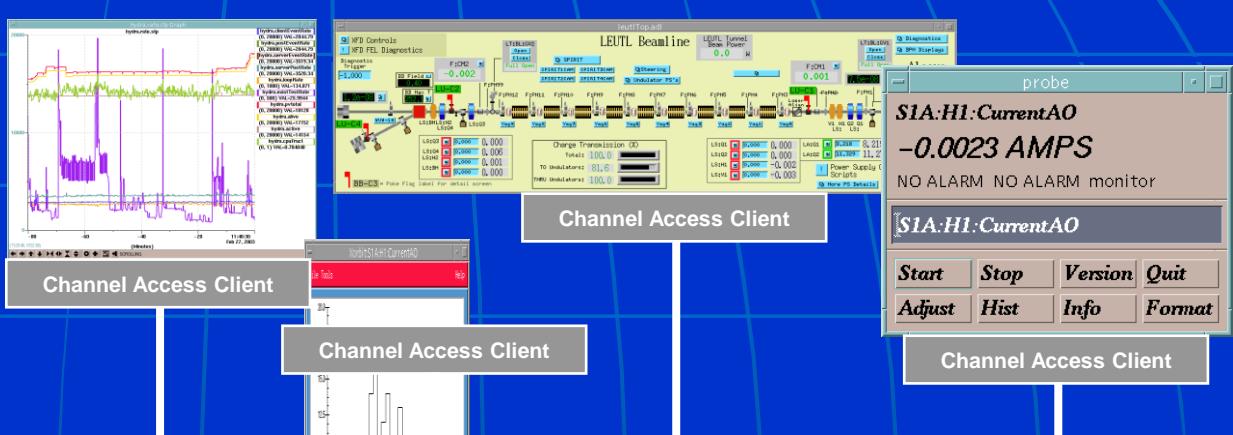
- Channel Access *clients* are programs that require access to **Process Variables** to carry out their purpose



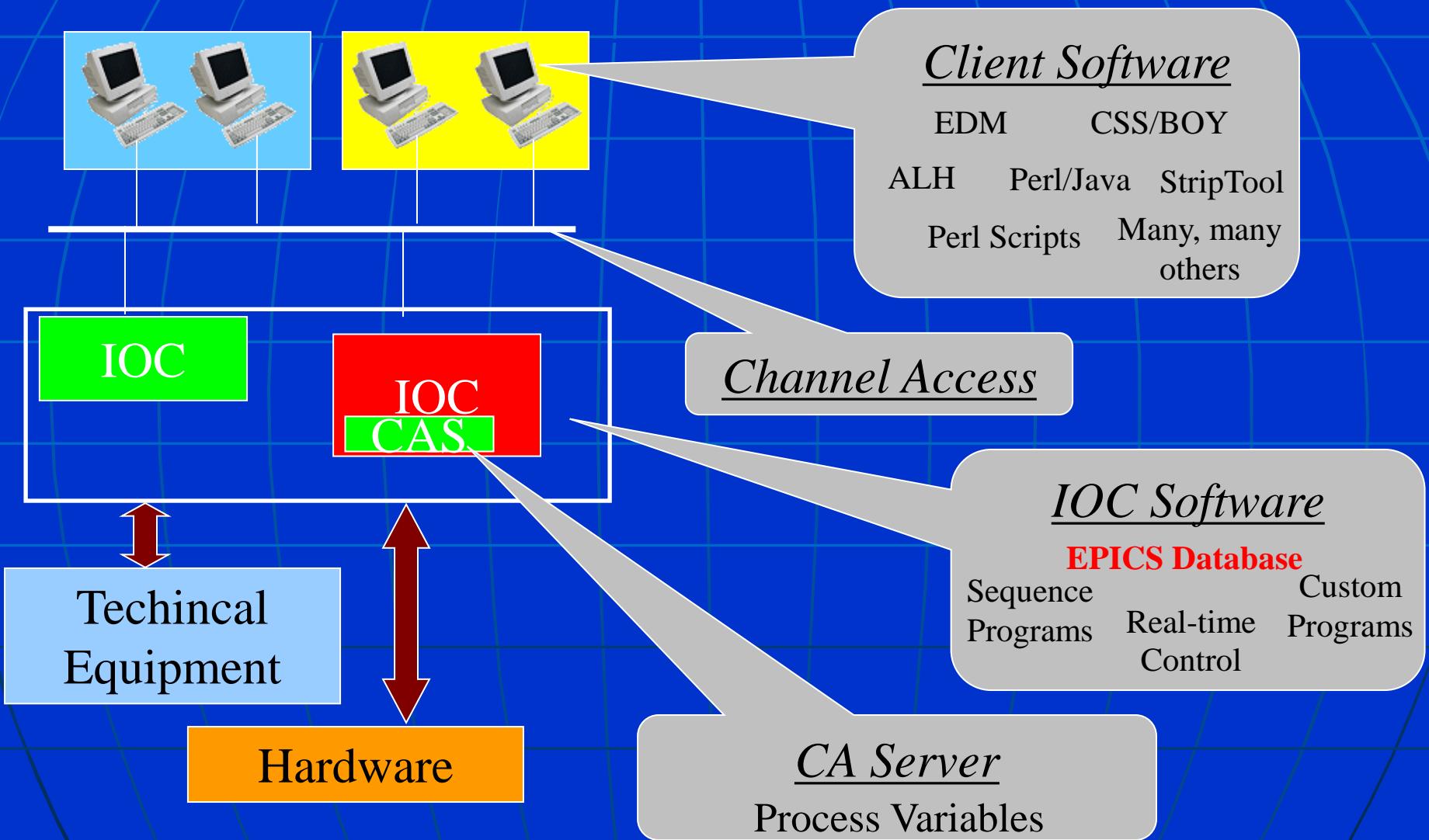
- The “service” that a Channel Access *server* provides is access to a **Process Variable***

* A Process Variable (PV) is a named item of data.

EPICS at a real facility



A typical EPICS Control System



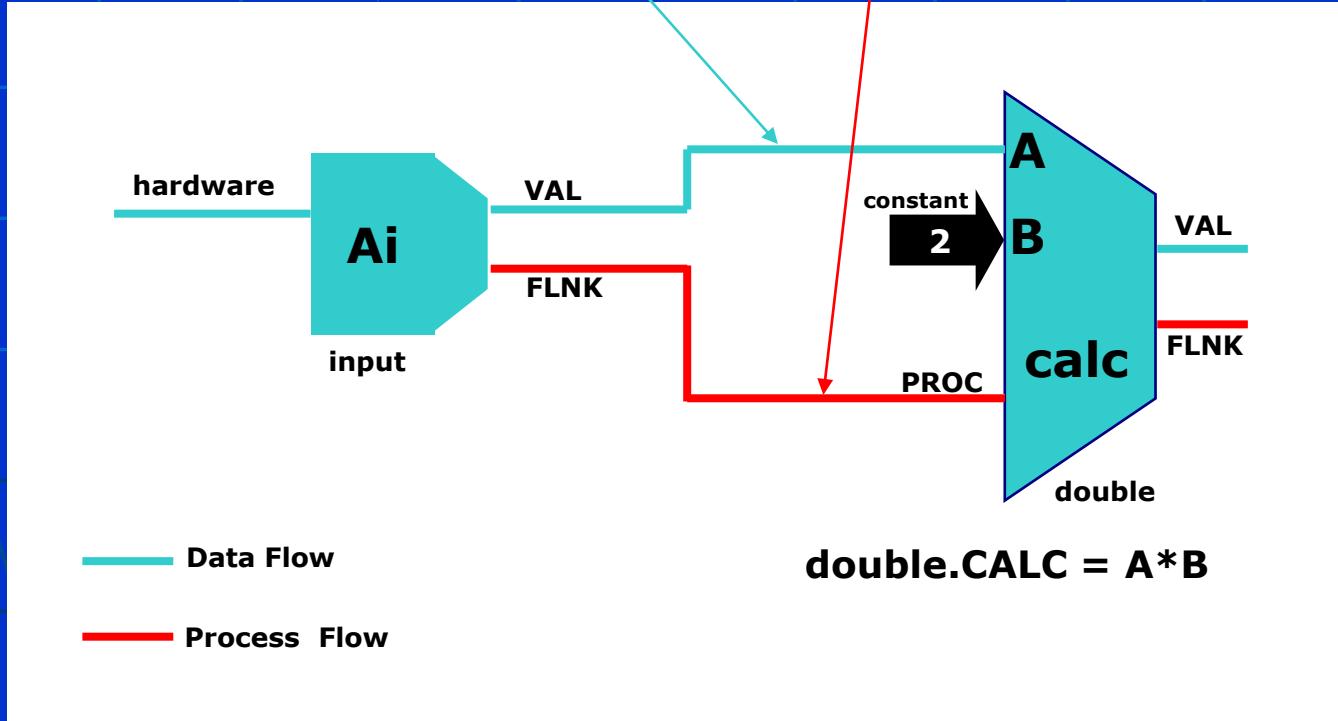
EPICS IOC Database

The EPICS database is the heart of the control system:

- Contains EPICS Records (of specific functionality)
- Application specific & memory resident
- Built by creating and linking many EPICS records together
- Deterministic: it runs in real-time, either:
 - synchronously (periodically driven)
 - asynchronously (event driven)
- Channel Access Security allows access control (R/W) based on user, location
- Able to simulate missing hardware

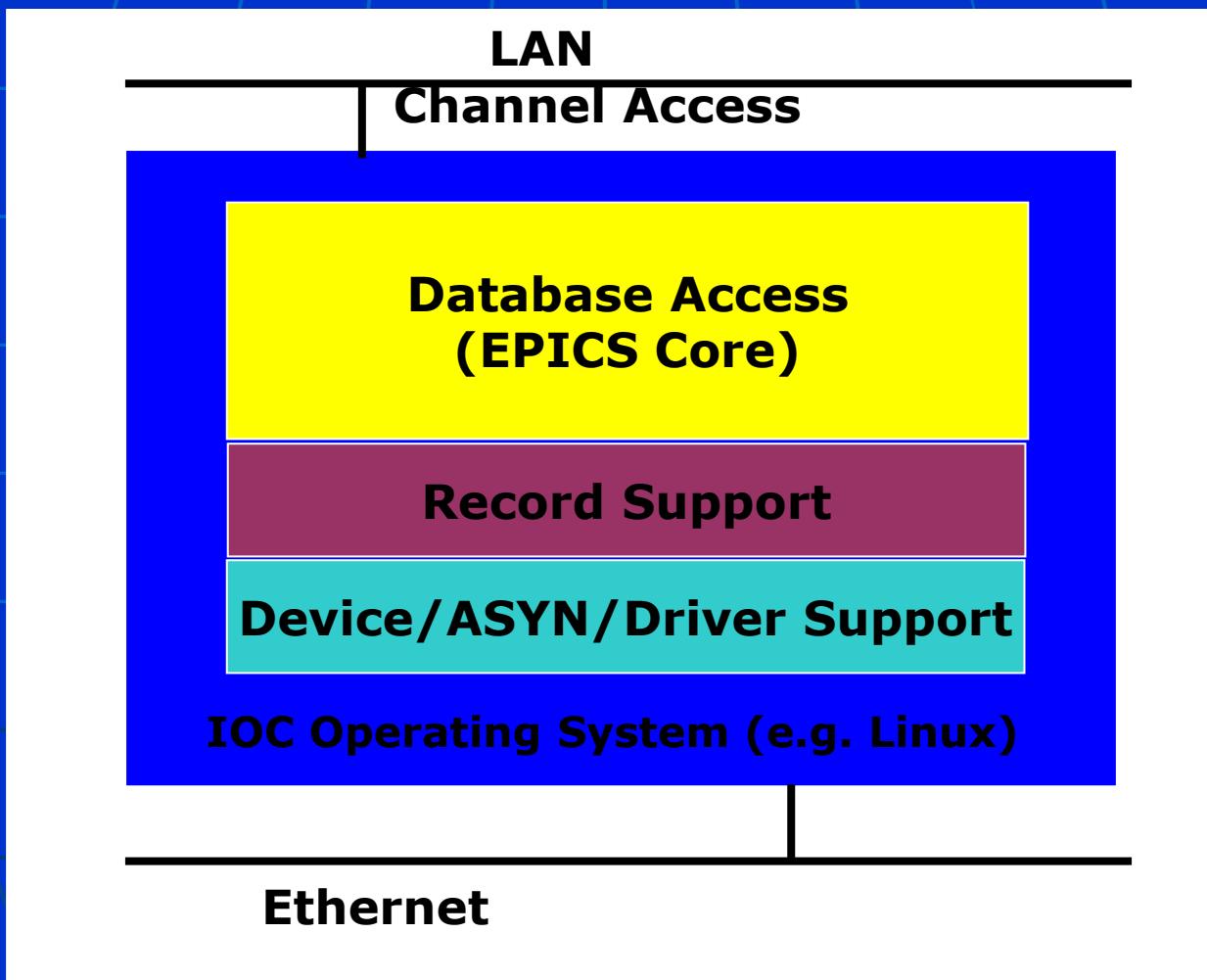
EPICS Database Example

Records in a database may be linked for purposes of data and processing flow





IOC Internals Overview





EPICS Record Types

The following 25 standard record types exist in EPICS Base

I/O	Signal Processing	Data Storage	Logic/Control
Ai	Calc	Stringin	Dfanout
Ao	Calcout	Stringout	Fanout
Bi	Sub	Waveform	Event
Bo	aSub	SubArray	Select
Mbbi		Aai	Sequence
MbbiDirect		Aao	
Mbbo			
MbboDirect			
Longin			
Longout			

EPICS Record Support

- Each record type has a template describing the processing it performs
- A database generally contains several instances of a record type

Generally, Record support will perform some combination of the following:

- I/O If record is an I/O type, it will read from or write to hardware via Device Support.
- Conversion Conversion of raw data to user defined units (with smoothing, scaling if configured).
- Alarms Check for and raise alarms (hihi, high, low, lolo).
- Monitor Post monitors on data values (which causes client callbacks).
- Link Cause processing of related records.

Record Fields

A record contains fields. Each field has a name abbreviated to no more than **4 upper case letters**.

A Process Variable (PV) is a field of a record which holds data. It is written as the name of the record followed by “.” and then the field of the record, specified like this: **<record name>.<field name>**

If no field name is given, the .VAL field is defaulted

All record types include a core set of fields which are needed for basic record processing, such as:

- SCAN The record's method of scanning
 - VAL The record's value
 - FLNK The record's processing link to other records

Input records (e.g. Analogue Input) always have:

- DTYP Device Support Entry Table (set of functions)
 - INP Link containing the hardware address to read

Output records (e.g. Binary Output) always have:

- DTYP Device Support Entry Table (set of functions)
 - OUT Link containing the hardware address to write

Record Processing (Scanning)

An EPICS record can be processed in 4 ways:

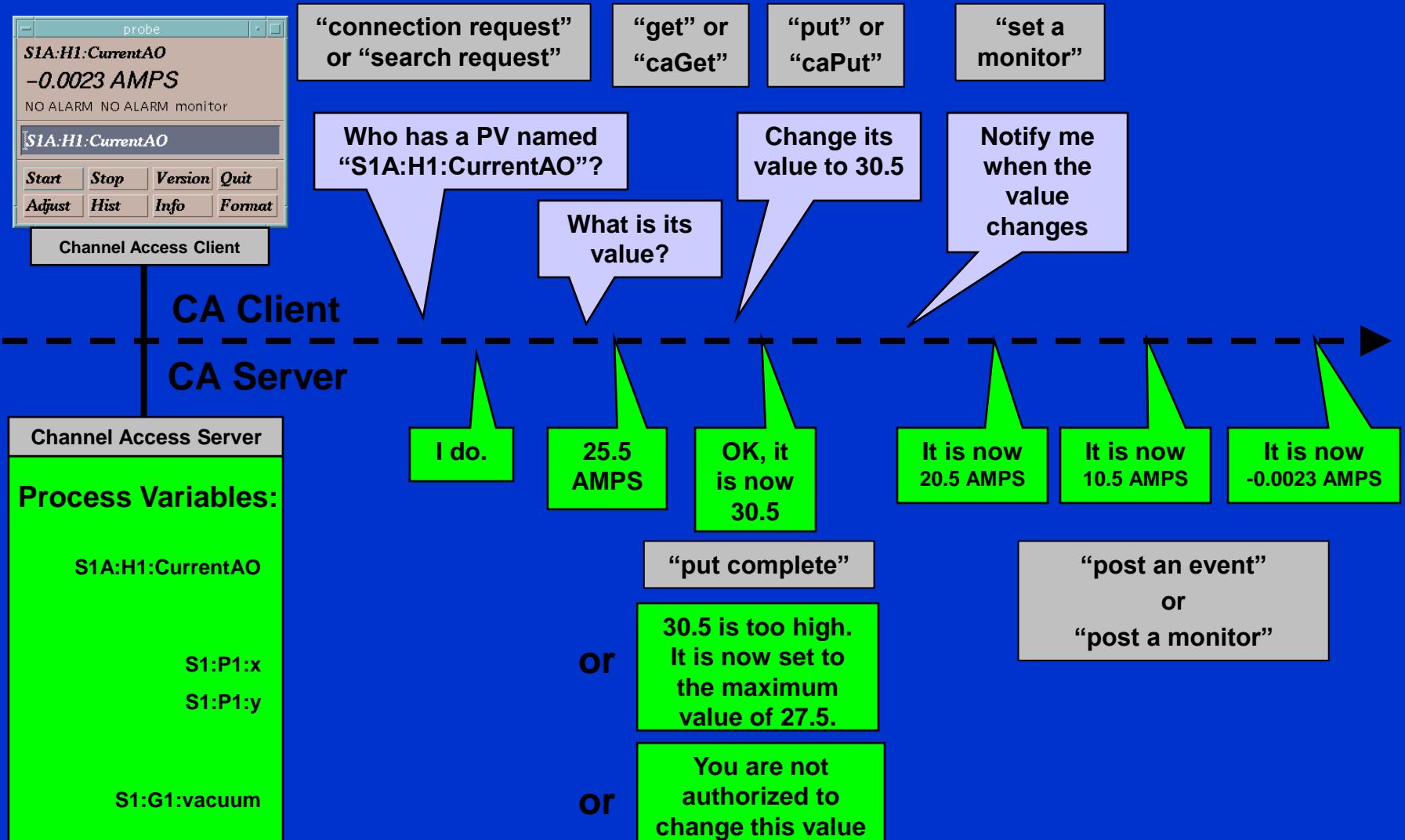
1. Periodic Scan There are 7 standard scan rates in EPICS ranging from 0.1 seconds to 10 seconds. This can be extended.
2. I/O Interrupt Processed by a hardware interrupt
3. Event Scan Processed as a result of a data “event” being posted in the database. Software interrupt.
4. Passive If a record is “Passive” it will only process if written to by another record or over channel access.

Within the IOC, records are processed by a scan task (or *thread*). There is a separate task for each scan period and scan type.

Channel Access

- Channel Access is the "backbone" of EPICS. It provides connections between the IOC database and external systems.
- Channel Access is based on a Client/Server model. Each IOC runs its own local Channel Access Server task(s).
- Channel Access runs over the network
 - using TCP/IP for reliable data transport
 - UDP/IP for connection management and name resolution
 - Generally, 3 operations that you can do: "get", "put" and "monitor".
- Originally written in "C" by Jeff Hill at Los Alamos in the late '80's.
- Native Java version was produced in 2005 by Matej Sekoranja (Cosylab).

Channel Access in One Slide



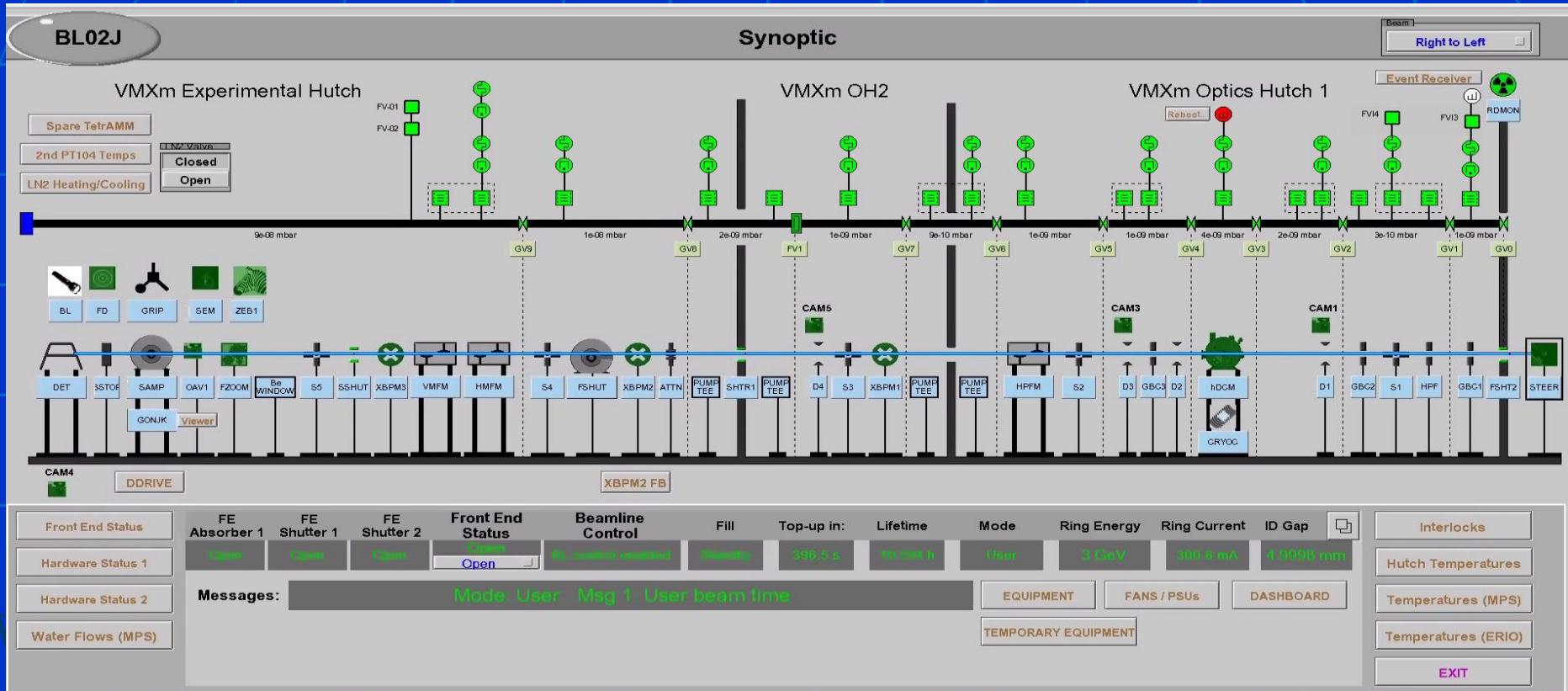


Some Operator Interface Tools

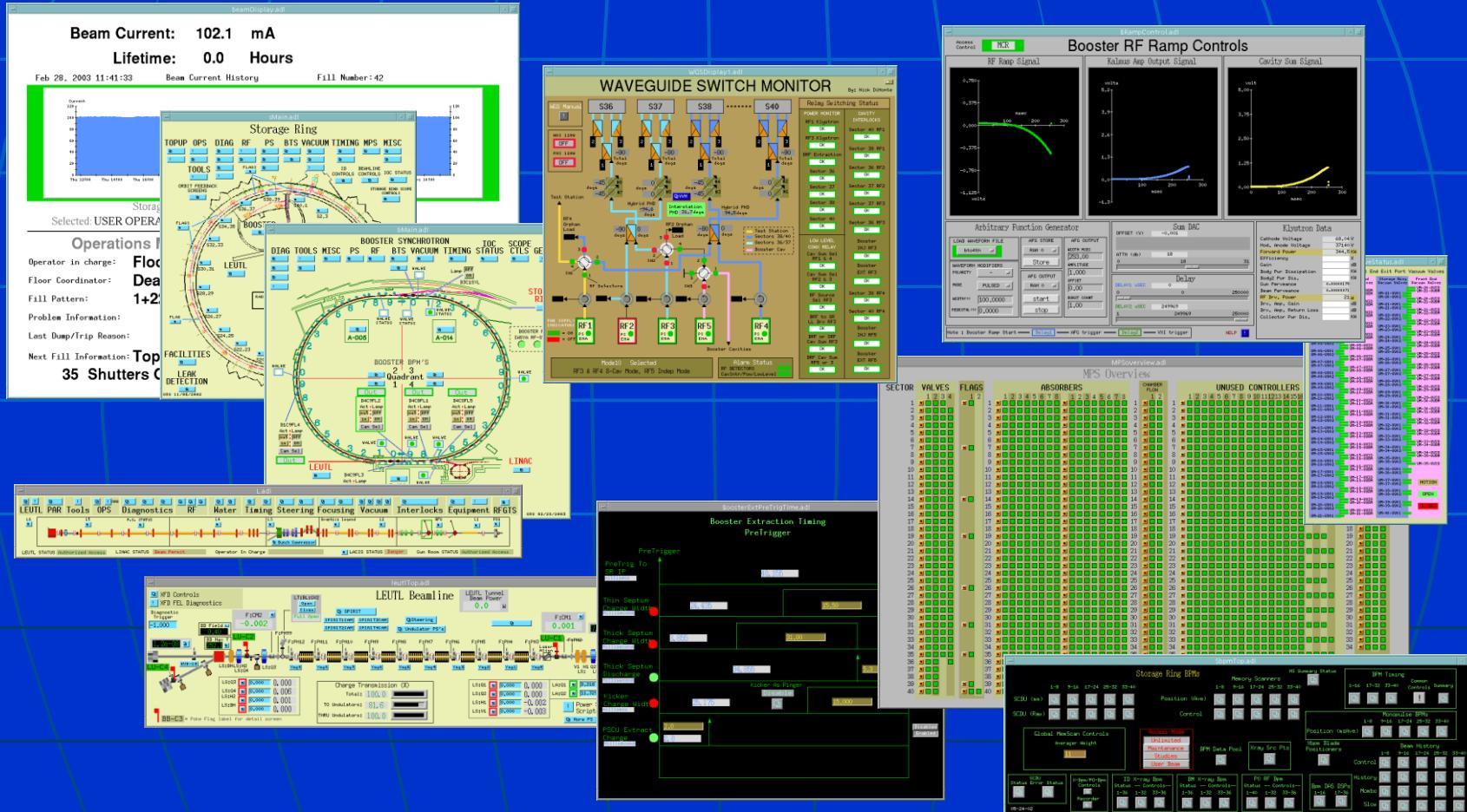
These are all host-based standalone utilities using Channel Access to configure or monitor the system. They are configured using interactive tools or files.

EDM	Extensible Display Manager GUI for control/monitoring (Diamond)
MEDM	Motif-based Display Manager (APS)
EPICS Qt	EPICS CA widgets for Qt GUI framework (Australia)
CS-Studio "RCP"	Original Control System Studio
CS-Studio "Phoebus"	"The latest"! No longer depends on Eclipse (SNS)
StripTool	Plots data as a strip-chart
ALH	Alarm Handler: graphical display of alarm tree
Archive Appliance	Archives and displays data

Example Screens: EDM



Example Screens: MEDM



Example Screens: CSS/BOY

CODAC SysSTATUS

- Not ready
- Ready
- Start of pulse sequence
- Wait for systems initialised
- Pre-pulse checks
- Final preparation
- Pulse
- After pulse checks

Stop 

Exclude 

FAST PLANT SYSTEM CONTROLLER - OPERATOR INTERFACE - MAIN MENU

PULSE

Pulse N.: 1390
Pulse Time: 26.0

TIMING

System Time: 2011/11/03 15:12:34
Countdown: 0.0

HARDWARE MONITOR

Temperatures

FTM TEMP #1	FTM TEMP #2	FTM TEMP #3
27.16	31.68	24.2

Fan Tray Speeds

Alarm Indicator LED:  2145

Fan#1 Speed (RPM): 2178E3

Fan#2 Speed (RPM): 2.178E3

Fan#3 Speed (RPM): 2145

°C

100
50
0
100
50
0
100
50
0



Entity: Instituto Plasmas Fusao Nuclear
Address: Instituto Superior Tecnico Av. Rovisco Pais 1049-001 Lisboa Portugal
Project: ITER - Fast Plant System Controller
Type: Operator Interface
Version: 0.0.1Beta
Revision Date: 02/11/2011

Copyright (c) 2011

MARTE SysSTATUS

Execute 

- Off
- Not ready
- Ready
- Initialising
- Initialised
- Executing
- Post pulse

Stop 

Channel-A

Voltage / Volts

-1.4396E6
-1.5E6
-1.6E6
-1.7E6
-1.8E6
-1.9E6
-2E6
-2.1E6
-2.1607E6

Time / s

4.3278E7 4.4E7 4.46E7 4.52E7 4.58E7 4.64E7 4.7E7 4.76E7 4.8278E7

Error (Volts): -2,160,745.000 volt
Voltage (Volts): -2,160,745.000 volt

MARTE FrameWork Cycle TIMING

Cycle Time: 0.0001 s
Calculation Time: 0.0001 s

LoopTime

Time (us)

1000
800
600
400
200
0

4.8163E7 4.82E7 4.824E7 4.828E7 4.8324E7

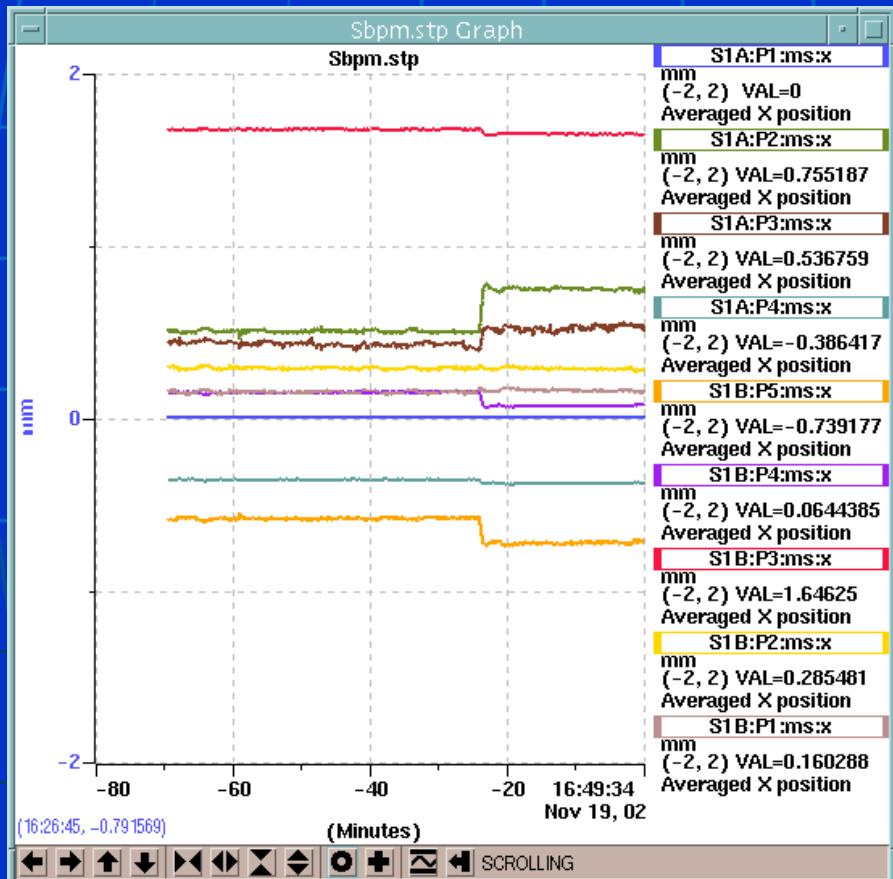
MARTE Time

Sync Data Network Transfer MONITOR

SDA Rate: 0.00 MB/s

Database

Example Screens: StripTool



Database Configuration Tools

These are all host-based standalone utilities used to define and create an EPICS database:

VDCT

Visual DCT. Java-based visual design tool

Capfast

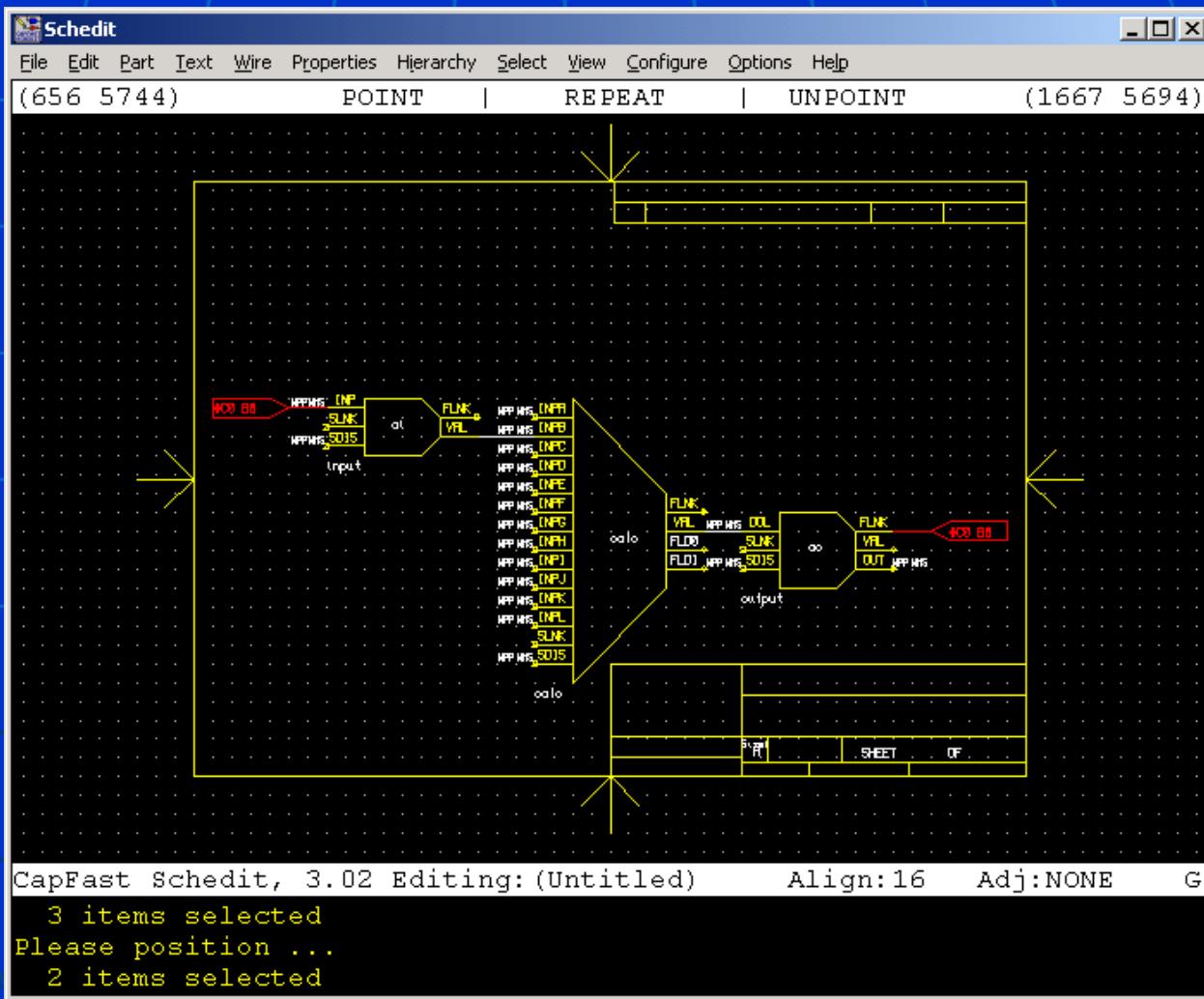
A proprietary (Phase 3 Inc.) electronics layout tool used to generate EPICS databases via hierarchical schematics

TDCT

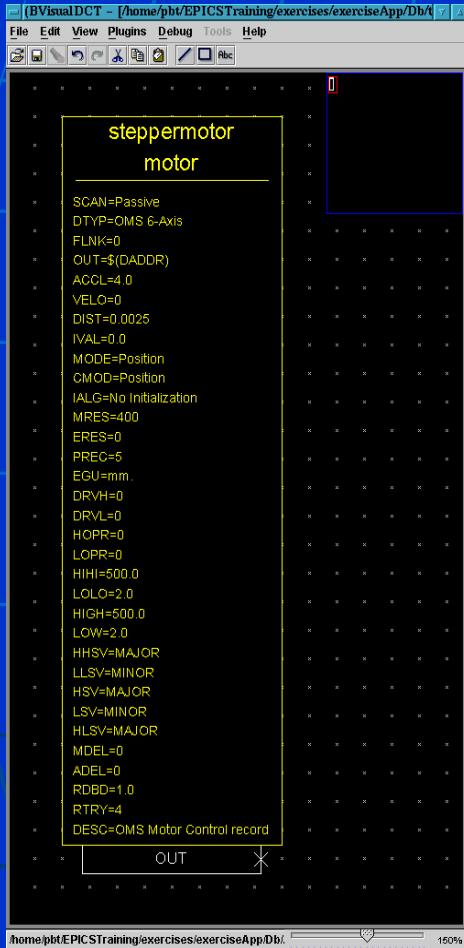
Triumf Database Configuration Tool. Java-based, Capfast compatible

The EPICS database file (.db file) has a straightforward text format and so can be generated or modified via other means including scripts and text editors (vi). But you cannot visualise links!

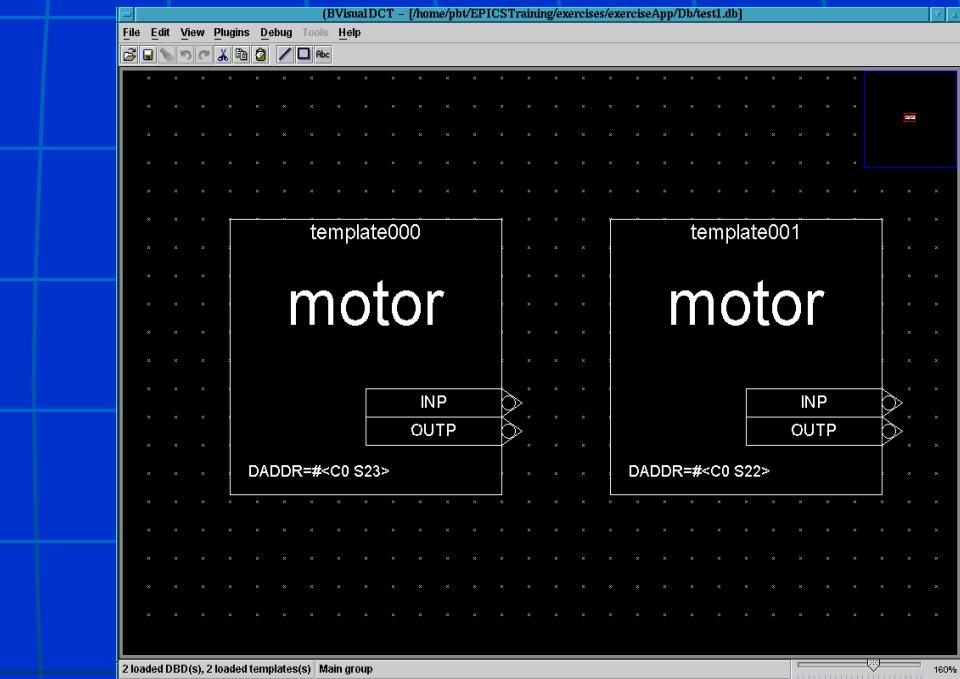
Capfast & TDCT



Configuration Tools: Visual DCT



Record



Templates

Channel Access Interfaces

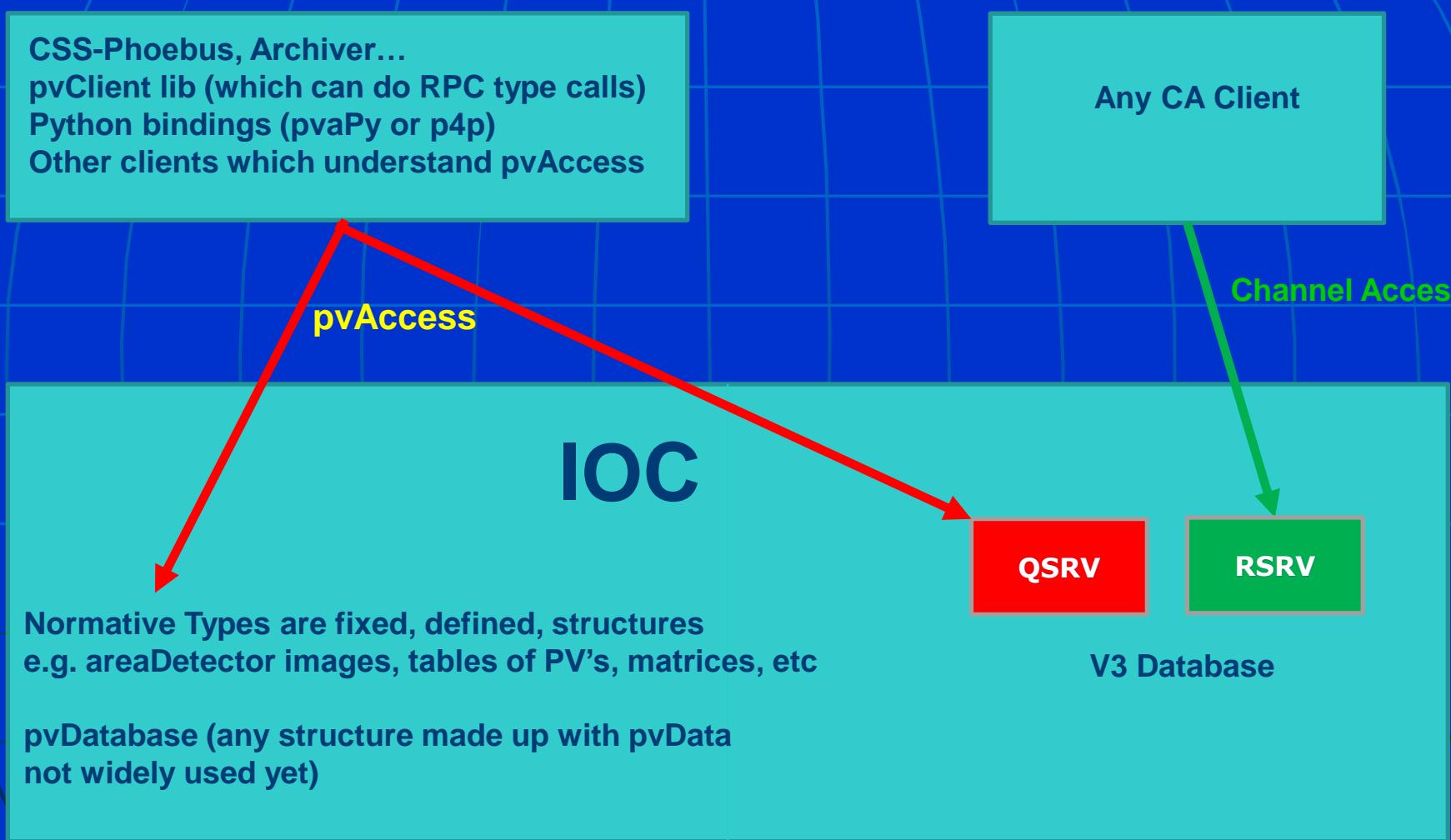
Many general-purpose programming languages as well as commercial and public-domain packages have been enabled with a Channel Access capability, including:

- | | |
|---------------|---------------------------------------|
| ➤ C | Standard Programming Languages |
| ➤ Java | Programming Language |
| ➤ Perl | Scripting Language |
| ➤ Python | Scripting Language |
| ➤ LabVIEW | Virtual Instrument Builder |
| ➤ Matlab | Numerical Computing |
| ➤ Unix Shells | Scripting languages |
| ➤ Mathematica | Mathematics/Modelling |
| ➤ Tcl/Tk | Interpreter with GUI |
| ➤ IDL/PV-Wave | Presentation Graphics |

The Future - EPICS 7

- Started with EPICS Version 4. Idea was to:
 - Add support for “structured” data in an EPICS database, not just simple types or arrays
 - Develop a new wire protocol called “pvAccess” to access these data
 - Add a “Remote Procedure Call” processing pattern, supporting passing arguments within the context of asynchronous record processing.
- EPICS Version 4 + EPICS Version 3 = EPICS 7.
- When you download EPICS 7, you get EPICS 3.16 Base plus the facilities developed in the version 4 project.

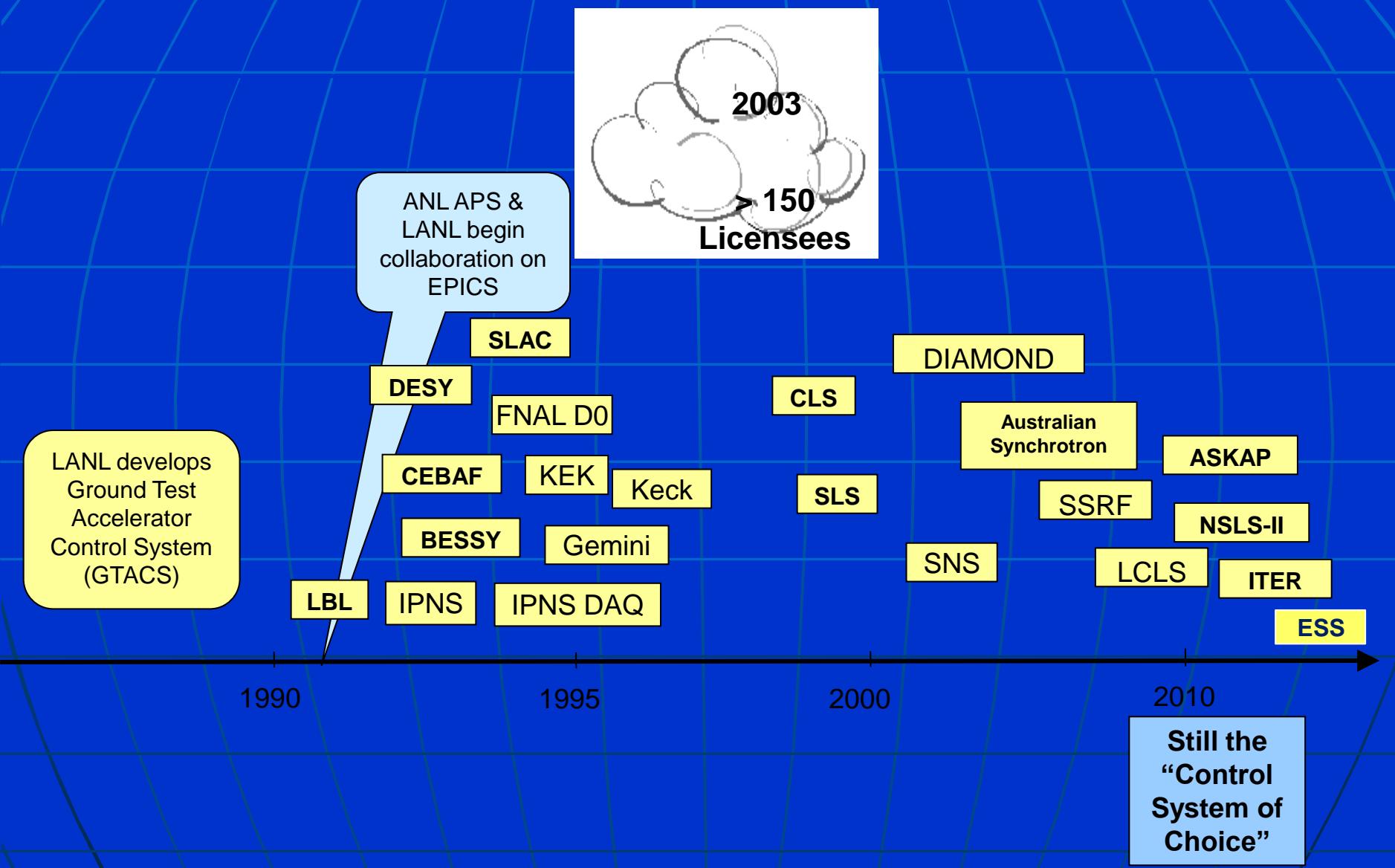
The Future - EPICS 7



Ten good things about EPICS

1. It's free
2. It's Open Source
3. There are lots of users running large, well-tested systems
4. All a client needs to know to access data is a PV name
5. You can pick the best tools out there ...
6. ... or build your own
7. The boring stuff is already done
8. There's lots of free advice available via email/web
9. A good contribution becomes internationally known
10. By following a few simple rules, you get a lot for free

Growth of EPICS 1990-2015



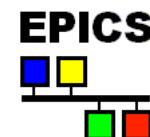
makeBaseApp

EPICS

Application Development Environment

Modified for the Diamond
environment

Andy Foster



What is makeBaseApp?

- **makeBaseApp** is the name of the standard EPICS build environment, distributed with EPICS base
 - Sets up standard application directory structure using a specified *template*
 - Sites create their own templates
 - implemented with perl script: `makeBaseApp.pl`

What we need

- Database files (“db” files)
- OPerator Interface files (OPI files - GUI screens)
 - GUI startup script (bash script to start GUI)
- Program files
 - device drivers, device support, sequence programs, subroutine record programs
- IOC startup files (what to load in the IOC)
 - Also called “IOC boot scripts”
- Configuration files
 - e.g. which EPICS release?

Setup directory for examples...

Start new terminal

mkdir course

cd course

mkdir scanner (EtherCAT related)

mkdir ioc

Let's run makeBaseApp - scanner

```
cd scanner
```

(next command all on **one** line)

```
makeBaseApp.pl -t scanner -a linux-x86_64
```

```
-T
```

```
/dls_sw/work/R3.14.12.7/support/ajf67/EPICSTemplates/DLS_EtherCAT  
scanner
```

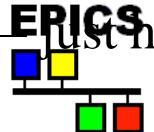
(next command all on **one** line)

```
makeBaseApp.pl -i -t scanner -a linux-x86_64
```

```
-T
```

```
/dls_sw/work/R3.14.12.7/support/ajf67/EPICSTemplates/DLS_EtherCAT  
scanner
```

(you will be prompted for: Which application should the IOC boot? **Just hit return**)



Let's run makeBaseApp - ioc

```
cd ..ioc
```

(next command all on **one** line)

```
makeBaseApp.pl -t dls -a linux-x86_64
```

```
-T
```

```
/dls_sw/work/R3.14.12.7/support/ajf67/EPICSTemplates/DLS_EtherC  
AT exercise
```

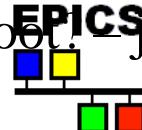
(next command all on **one** line)

```
makeBaseApp.pl -i -t dls -a linux-x86_64
```

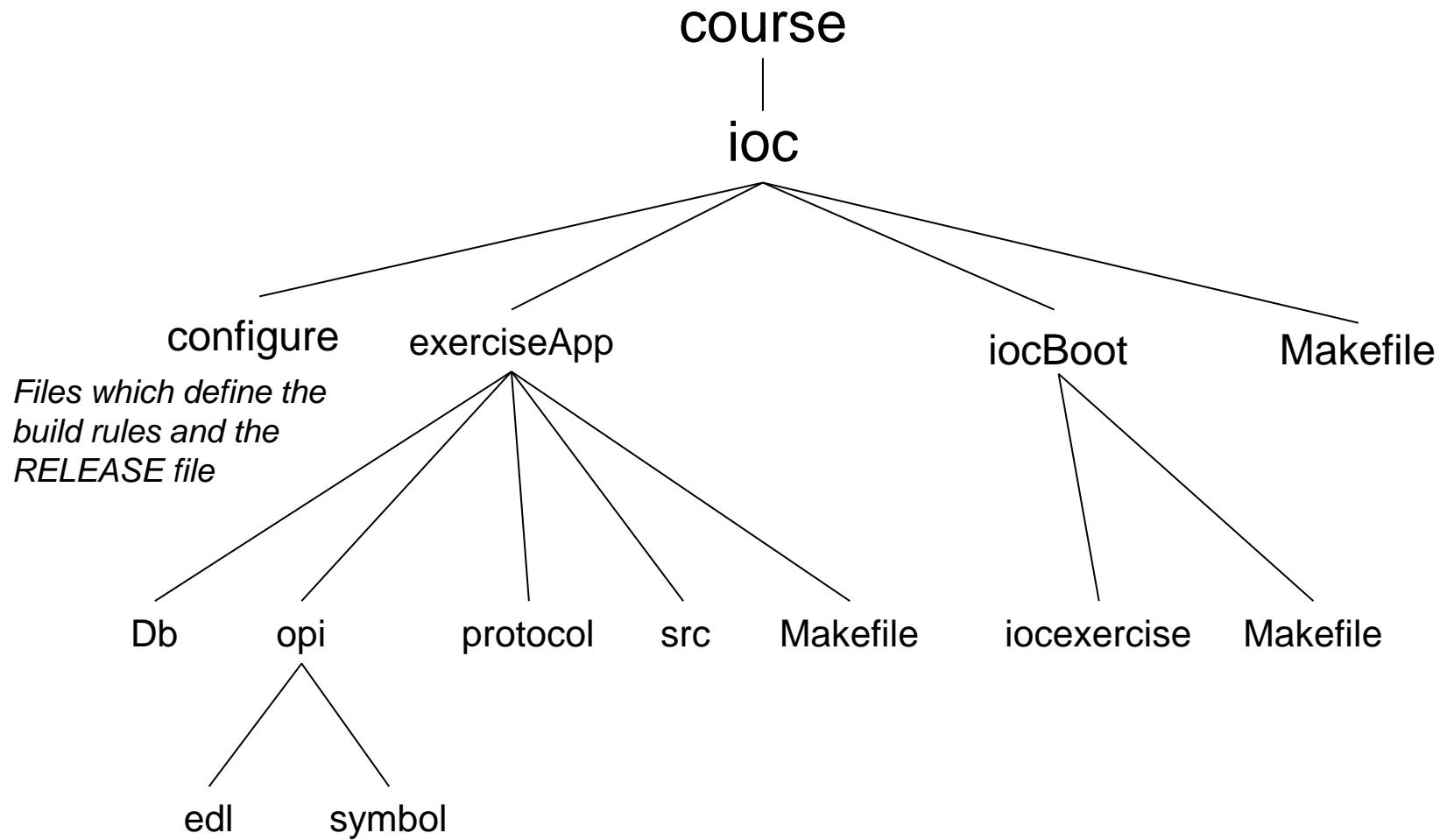
```
-T
```

```
/dls_sw/work/R3.14.12.7/support/ajf67/EPICSTemplates/DLS_Etherc  
at exercise
```

(you will be prompted for: Which application should the IOC boot? Just hit return)



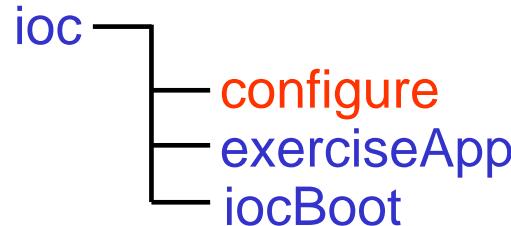
Application Directory Structure



Application Directory Structure – source files

- The application directory structure distinguishes between *source* files and *built* files
- What we have just created is a directory structure to hold the source files
 - Source files -> Source code repository (CVS, Subversion, **Git**, Mercurial...)
- *built* files are generated by “make”
- Makefile at each level of the directory structure
- *built* files do NOT go into the repository!
- **Do not edit the *built* files, edit the *source* files and type “make”!**

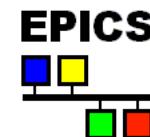
Configuration Files



- **configure** subdirectory

- Rules for how this application builds and compiles
- The **RELEASE** file defines the version of EPICS base and which modules should be searched when building the IOC.

SUPPORT= /dls_sw/prod/R3.14.12.7/support
WORK= /dls_sw/work/R3.14.12.7/support
ASYN= \$(SUPPORT)/asyn/4-41
MOTOR= \$(SUPPORT)/motor/7-0dls7
SNCSEQ= \$(SUPPORT)/seq/2-2-5dls1
UTILITY= \$(SUPPORT)/utility/dls2-19
PMAC= \$(SUPPORT)/pmac/2-4-23
STREAM = \$(SUPPORT)/streamDevice/2-5dls13
EPICS_BASE= /dls_sw/epics/R3.14.12.7/base

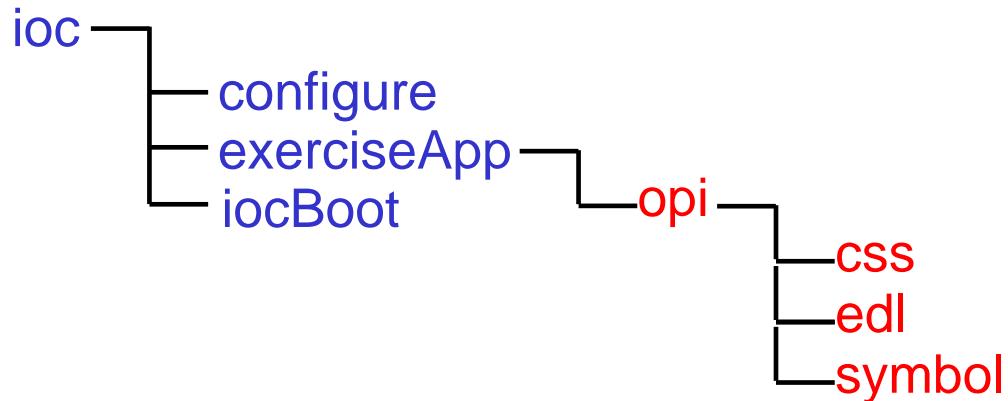


Source files – Db directory



- Database files
 - contains “db” files
 - contains “template” files: “*.template”
 - contains “substitutions” files: “*.substitutions”

Source files – opi directory



- Operator Interface Directory (GUI screens)
 - Contains further sub-directories:
 - `edl` (EDM display lists – ASCII files)
 - Also contains bash script, “`startgui1`” to start the GUI
 - `symbol` (EDM symbols – see EDM talk)
 - `css` (does not exist yet but this is where it should live. Control System Studio GUIs)

Source files – protocol directory



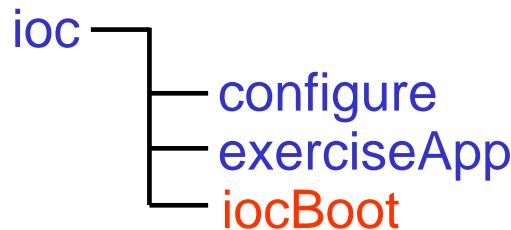
- Stream Device Protocol Files
 - Define ASCII protocols for communicating with serial devices
 - Lots of different serial devices!

Source Files – src directory



- C files (**.c**)
- C++ files (**.cpp**)
- State Notation Language files (**.stt**)
- Scripts
- Database Definition files (**.dbd**)

Source Files – iocBoot directory

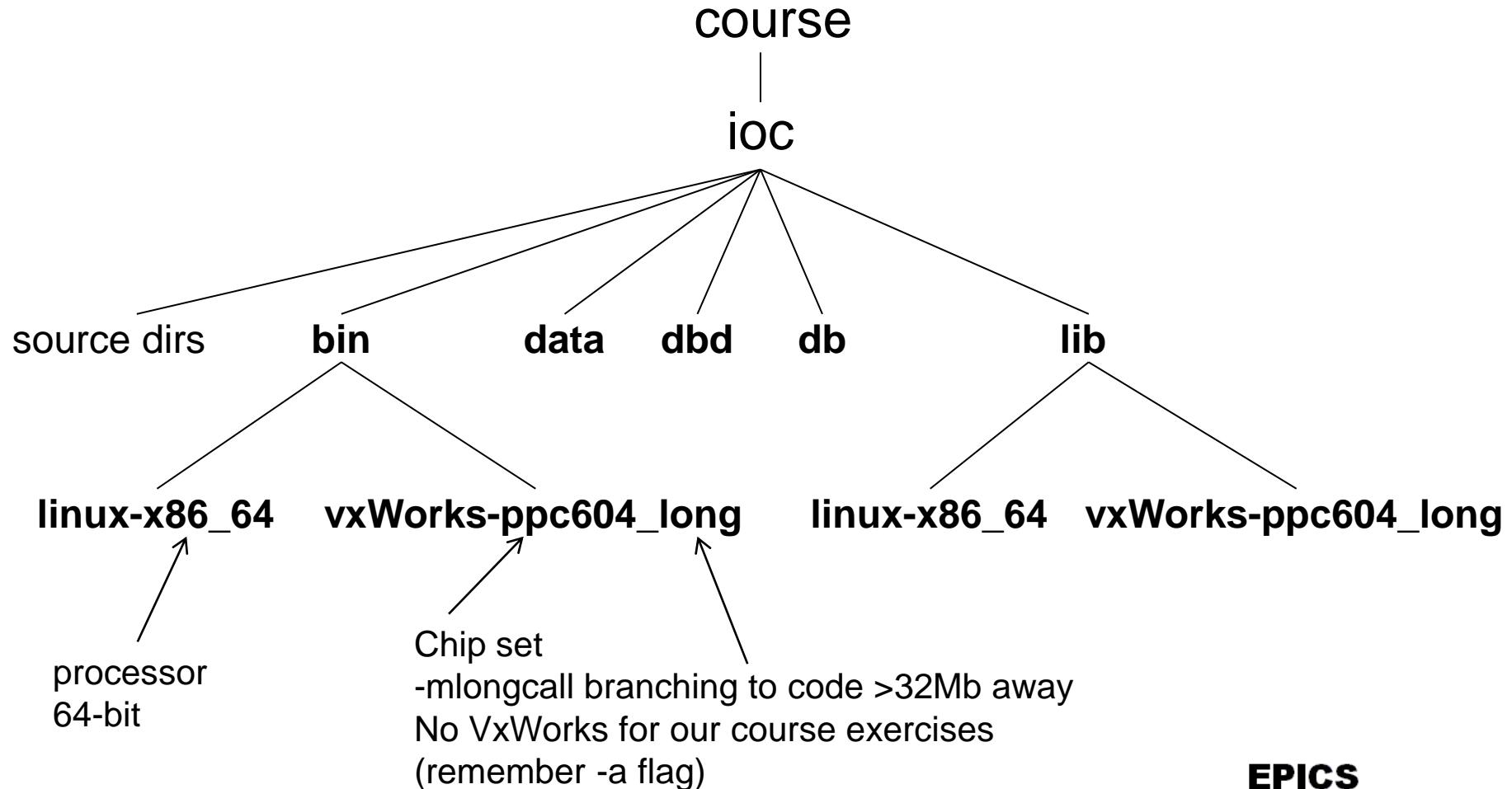


- This directory is created by the “-i” option to `makeBaseApp.pl`
 - Contains the source file from which to generate the IOC boot script: e.g. `stexercise1.src`.
 - Contains a “Makefile”
 - **When you want to make changes to the boot script, edit the src file and type “make”!**

Built or Generated Files

- In “course/ioc”, please type “make”
- There is a Makefile at each level in the directory structure, so the build propagates down from the level you start at
- After “make”, the generated directories live directly below the top-level directory. These are:
 - bin
 - db
 - dbd
 - lib (not for our exercises, we are not building support libraries)
 - data

Directory structure showing generated directories



Generated Files – db directory

- Database files (*.db)
 - Files in here are produced as a result of “make” in the source “Db” directory, usually as a result of combining a template file and a substitutions file
 - See “Database II” presentation later!

Generated Files – dbd directory

- Database Definition files (*.dbd)
 - created by **make** in src
 - **make** creates a single *dbd* file for the IOC by combining individual *dbd* files together
 - The “dbd” file tells EPICS about record properties, which device supports and drivers are available to this IOC.
 - See “Database II” presentation later!

Generated Files – bin directory

- Binary files
 - For multiple architectures (we specified linux-x86_64)
 - Created by **make** within the **src** directory
 - compiled from “C” files, “C++” files and State Notation Language files (*.stt)
- IOC boot file compiled from the startup script in *iocBoot/iocexercise* (replaces macros in startup script) e.g.

```
epicsEnvSet("STREAM_PROTOCOL_PATH","$(PMACUTIL)/data:$(PMACCOORD)/data")
```

Generated Files – lib directory

- Binary shared library files
 - for multiple architectures
 - Created by **make** from within **src** directory
 - We must specify that we are building a library in the Makefile

e.g. LIBRARY_IOC = pmacAsynMotorPort

pmacAsynMotorPort_SRCS = pmacController.cpp

pmacAsynMotorPort_SRCS = pmacAxis.cpp

...

(We are not using this make rule in the exercises)

Generated Files – data directory

- Architecture independent ASCII files, such as:
 - EDM display lists
 - Scripts
 - Stream device protocol files
 - Populated as a result of “make” in the “opi/edl” subdirectory, the “src” subdirectory and the “protocol” subdirectory

Diamond improvements

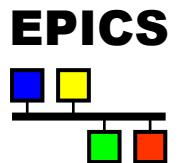
- The version of **makeBaseApp.pl** described here includes a lot of work by Diamond, on the build rules, to turn it into an environment which fully supports “source” and “built” files in the directory tree.
- The default version of **makeBaseApp.pl** distributed with EPICS does **NOT** do this! In particular:
 - IOC startup scripts are not built, but used in place.
 - Diamond can generate architecture dependent boot files from the same source file.
 - GUI screens (OPI files) are not built, but used in place.
 - There is no “data” directory.

makeBaseApp.pl: Command options

- a Set the architecture e.g. linux-x86_64
- b The EPICS base directory (default: use path of makeBaseApp.pl command)
- d Verbose output for debugging
- i Create an iocBoot area
- l List the available templates
- t The name of the template to use
- T The application template top-level directory

Reference

- EPICS Application Developer's Guide
(Chapter 4: EPICS Build Facility)
- <https://epics.anl.gov/base/R3-15/5-docs/AppDevGuide/AppDevGuide.html>
- <https://docs.epics-controls.org/en/latest/appdevguide/AppDevGuide.html>

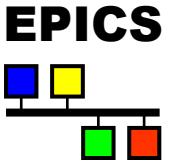


What is an EPICS Database?

Andrew Johnson
APS

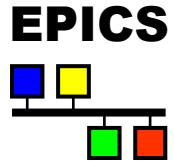
(modified by Philip Taylor & Andy Foster 2004-2009)

Outline



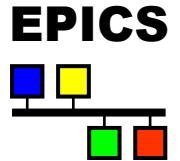
- Records
- Fields and field types
- Record Scanning
- Input and Output record types
- Hardware support
- Links
- Chaining Records together
- Protection mechanisms
- Alarms, deadbands, simulation and security

Database = Records + Fields + Links



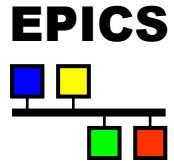
- A control system using EPICS will contain one or more IOCs
- Each IOC loads one or more Databases telling it what to do
- A Database is a collection of Records of various types
- A Record is an object with:
 - A unique name
 - A behaviour defined by its record type (class)
 - Controllable properties (fields)
 - Optional associated hardware I/O (device support)
 - Links to other records

Record Activity



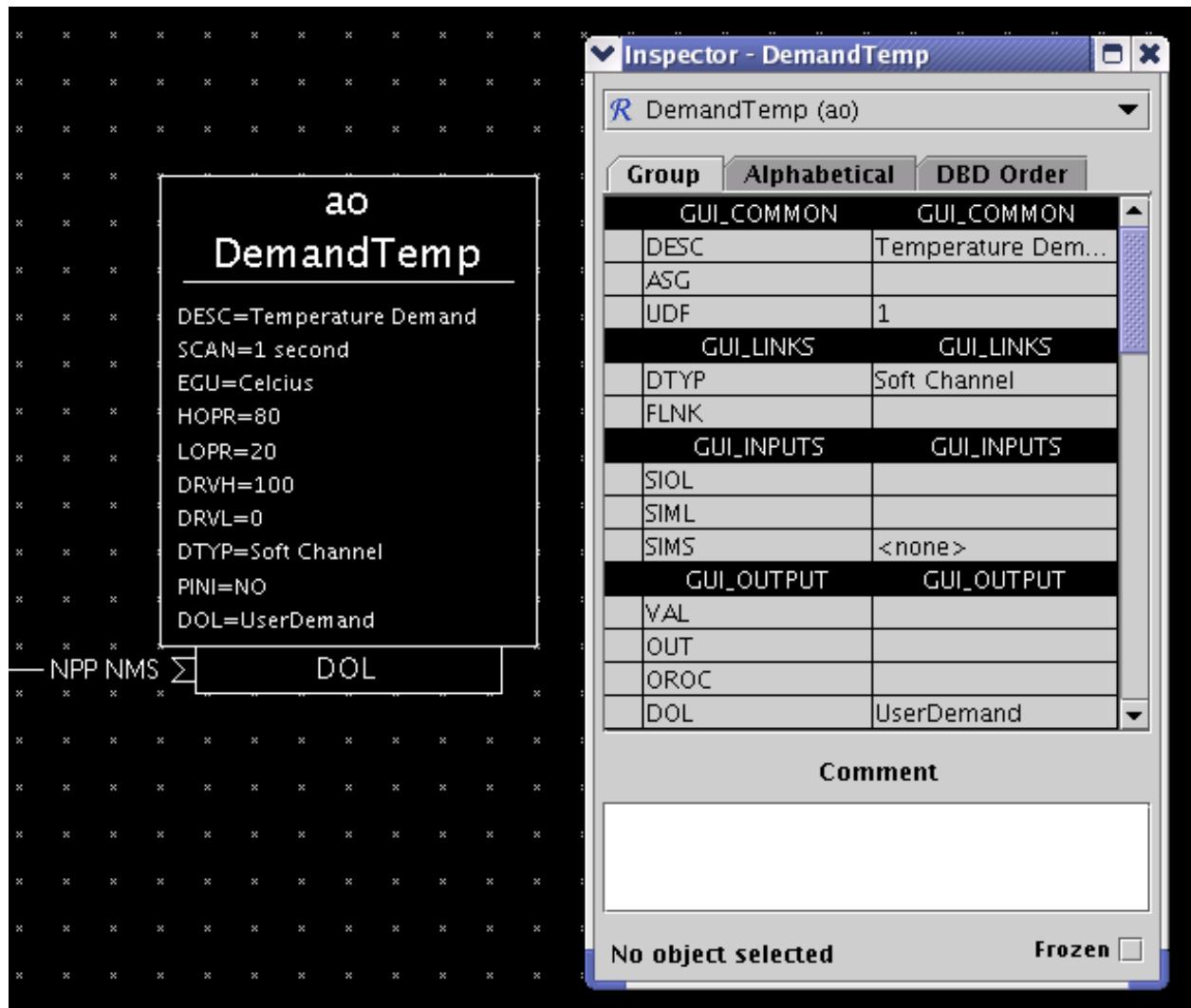
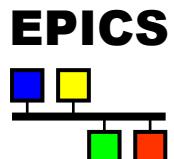
- Records are active – they can do things:
 - Get data from other records or from hardware
 - Put data to other records or to hardware
 - Perform calculations
 - Check values are in range & raise alarms
 - Wait for hardware signals (interrupts)
- What a record does depends upon its record type and the settings of its fields
- No action occurs unless a record is processed

How is a Record implemented?

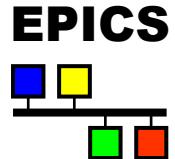


- A ‘C’ structure with both data storage and pointers to record type information
- A record definition within a database provides
 - Record name
 - The record’s type
 - Values for various fields
- A record type provides
 - Definitions of all the fields
 - Code which implements the record behaviour
- New record types can be added to an application as needed

Visual DCT view of a record



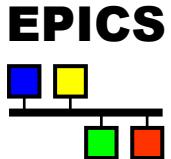
The IOC's view



The full .db file entry for an Analogue Output Record

```
record(ao, "DemandTemp") {  
    field(DESC, "Temperature")  
    field(ASG, "")  
    field(SCAN, "Passive")  
    field(PINI, "NO")  
    field(PHAS, "0")  
    field(EVNT, "0")  
    field(DTYP, "VMIC 4100")  
    field(DISV, "1")  
    field(SDIS, "")  
    field(DISS, "NO_ALARM")  
    field(PRIO, "LOW")  
    field(FLNK, "")  
    field(OUT, "#C0 S0")  
    field(OROC, "0.0e+00")  
    field(DOL, "")  
    field(OMSL, "supervisory")  
    field(OIF, "Full")  
    field(PREC, "1")  
    field(LINR, "NO CONVERSION")  
    field(EGUF, "100")  
    field(EGUL, "0")  
    field(EGU, "Celcius")  
  
    field(DRVH, "100")  
    field(DRVL, "0")  
    field(HOPR, "80")  
    field(LOPR, "10")  
    field(HIHI, "0.0e+00")  
    field(LOLO, "0.0e+00")  
    field(HIGH, "0.0e+00")  
    field(LOW, "0.0e+00")  
    field(HHSV, "NO_ALARM")  
    field(LLSV, "NO_ALARM")  
    field(HSV, "NO_ALARM")  
    field(LSV, "NO_ALARM")  
    field(HYST, "0.0e+00")  
    field(ADEL, "0.0e+00")  
    field(MDEL, "0.0e+00")  
    field(SIOL, "")  
    field(SIML, "")  
    field(SIMS, "NO_ALARM")  
    field(IVOA, "Continue normally")  
    field(IVOV, "0.0e+00")  
}
```

Fields are for...

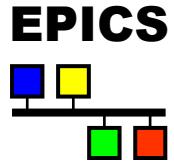


- Defining
 - What causes a record to process
 - Where to get/put data from/to
 - How to turn raw I/O data into a numeric engineering value
 - Limits indicating when to report an alarm
 - When to notify value changes to a client monitoring the record
- Holding run-time data
 - Input or output values
 - Alarm status, severity and acknowledgements
 - Processing timestamp

Field types

- Fields can contain
 - Integers
 - char, short or long
 - signed or unsigned
 - Floating-point numbers
 - float or double
 - Strings
 - max length 40 characters or less (CA restriction)
 - Long strings at 3.14.11 (add \$ to end of string or link fields forces it to be array of char)
 - Menu choices
 - select one from several strings
 - stored as a short integer
 - Links
 - to other records in this or other IOCs
 - to hardware signals (device support)
 - provide a means of getting or putting a value

All Records have these fields



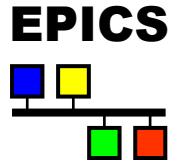
Design fields

NAME	60 character unique name (increased from 29 at 3.14.1)
DESC	40 character description string
SCAN	Scan mechanism
PHAS	Scan order (phase)
PINI	Process at startup?
SDIS	Scan disable input link
DISV	Scan disable value
DISS	Disabled severity
FLNK	Forward link

Run-time fields

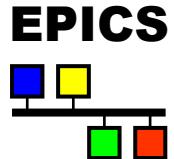
PROC	Force processing
PACT	Process active
STAT	Alarm status
SEVR	Alarm severity
TPRO	Trace processing
UDF	Set if record value undefined
TIME	Time when last processed

Record Scanning



- SCAN field is a menu choice from
 - Passive (default)
 - Periodic – 0.1 seconds .. 10 seconds
 - Hardware I/O Interrupt (only if the device supports this)
 - Soft event – EVNT field
- The number in the PHAS field allows processing order to be set within a scan
 - Records with PHAS=0 are processed first
 - Then those with PHAS=1 , PHAS=2 etc.
- The PINI field sets if the record is processed at startup (iocInit)
 - NO, YES
- PRIO field selects Low/Medium/High priority for Soft event and I/O Interrupts
- A record is also processed whenever any value is written to its PROC field

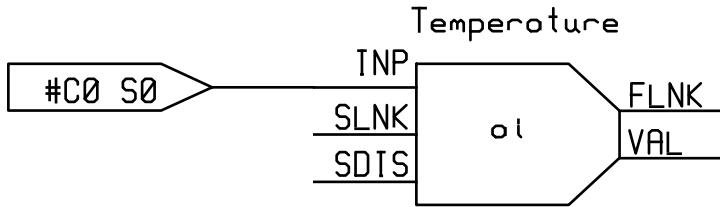
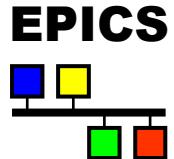
Input records often have these fields



INP	Input link
DTYP	Device type
RVAL	Raw data value
VAL	Engineering value
LOPR	Low operator range
HOPR	High operator range

- Analogue I/O records have these fields:
 - LINR Unit conversion control
 - ↳ No conversion, Linear, breakpoint tables...
 - EGUL Low engineering value
 - EGUF High engineering value
 - EGU Engineering unit string

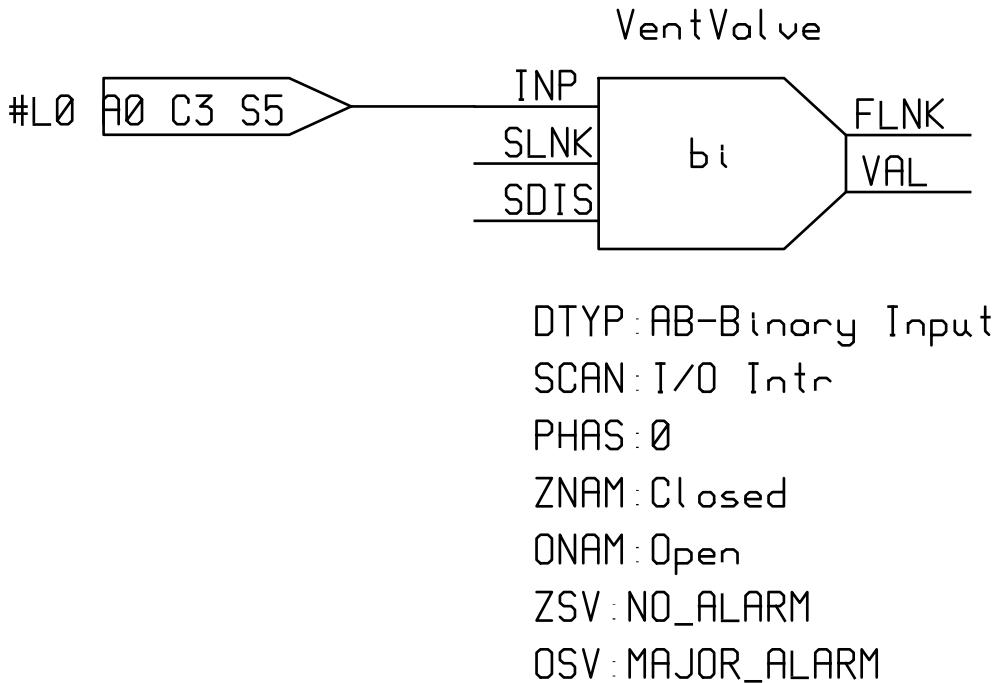
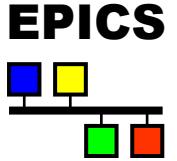
Periodic Input



DTYP XY566
SCAN 1 Second
PHAS 0
LTNR LTNEAR
EGUL 0
EGUF 120
EGU Celsius

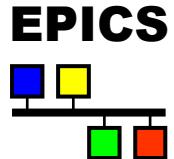
- Analogue Input “Temperature”
- Reads from the Xycom XY566 ADC Card 0 Signal 0
- Gets a new value every 0.1 seconds
- Data is converted from ADC range to 0..120 Celsius

Interrupt Input



- Binary Input “VentValve”
- Reads from Allen-Bradley TTL I/O Link 0, Adaptor 0, Card 3, Signal 5
- Processed whenever value changes
- 0 = “Closed”, 1 = “Open”
- Major alarm when valve open

Output records often have these fields

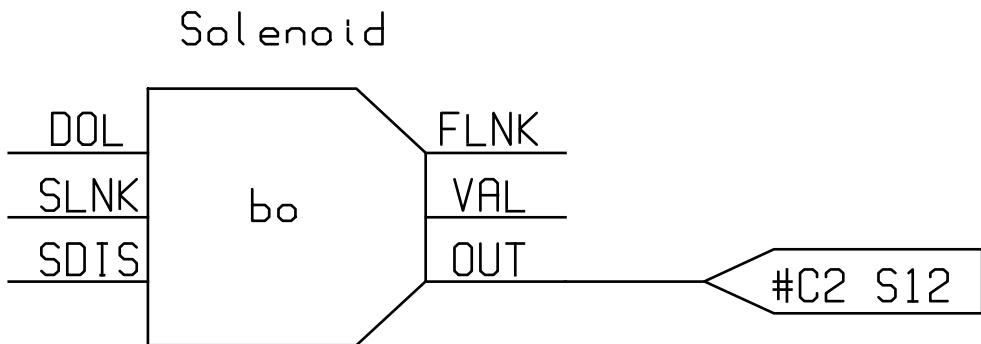
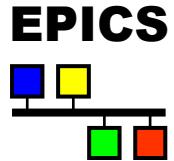


OUT	Output link
DOL	Input link to fetch output value
OMSL	Output mode select
↳	Supervisory, Closed Loop
LOPR	Low operator range
HOPR	High operator range

□ Analogue outputs also have these fields:

OROC	Output rate of change
OIF	Incremental or Full output
OVAL	Output value
DRVH	Drive high limit
DRVL	Drive low limit
IVOA	Invalid output action
IVOV	Invalid output value
RBV	Read-back value

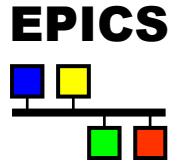
Passive Output



DTYP: XY220
SCAN: Passive
PHAS: 0
ZNAM: Locked
ONAM: Unlocked

- Binary Output “Solenoid”
- Controls Xycom XY220 Digital output Card 2 Signal 12
- Record’s SCAN field is set to Passive – it is processed according to the rules below

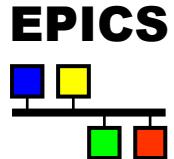
Links



A link is a type of field, and is one of

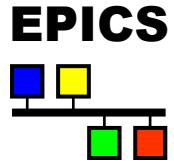
- Input link
 - Fetches data
- Output link
 - Writes data
- Forward link
 - Points to the record to be processed once this record finishes processing

Input and Output links may be...



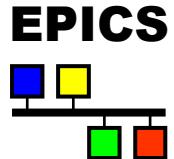
- Constant numeric value, eg:
 - 0
 - 3.1415926536
 - 1.6e-19
- Hardware link
 - A hardware I/O signal selector, the format of which depends on the device support layer
- Process Variable link – the name of a record, which at run-time is resolved into
 - Database link
 - Named record is in this IOC
 - Channel Access link
 - Named record not found in this IOC

Device Support



- Input and Output records are designed to perform hardware I/O via device support
- Device support is specified by the DTYP record
- Device support interprets the input/output link
- Soft device support allows use of DB/CA links
- 2 kinds of soft device support are provided:
 - Soft Channel
 - Get/Put VAL through link, no conversion
 - Raw Soft Channel
 - Get/Put VAL with data conversion

Database links



These comprise:

- The name of a record in this IOC
myDb:myRecord
- An optional field name
.VAL (default)
- Process Passive attribute
 - .NPP (default on INP and OUT links)
 - .PP (default for forward links)
- Maximize Severity attribute
 - .NMS (default)
 - .MS (Maximize Severity)
 - .MSI (Maximize Severity Invalid only)
 - .MSS (Maximize Severity and Status)

For example:

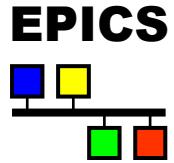
M1:current.RBV .NPP .MS

Channel Access Links

- Specified like a database link
- Name is not a record found in this IOC
- Use Channel Access protocol to communicate with remote IOC
- May include a field name (default .VAL)
- Input CA links don't cause destination record processing
- Output CA links obey "pp" attribute of destination field

Channel Access

Link Flags



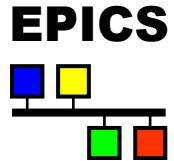
- Additional database link flags:
 - .CP Input link only. Use Channel Access. Process this record when destination value changes. Essentially, places a monitor on a field in a different record. Very useful to get an event-driven record.
 - .CA Forces a “local” database link to use Channel Access. Applies to both input and output links.

Forward Links

- Usually a database link referring to a record in same IOC
- Destination record must have
SCAN = Passive
for it to be processed
- Does not pass a value, just causes subsequent processing
- Channel Access forward links are possible by naming the PROC field of a remote record as the destination of the link

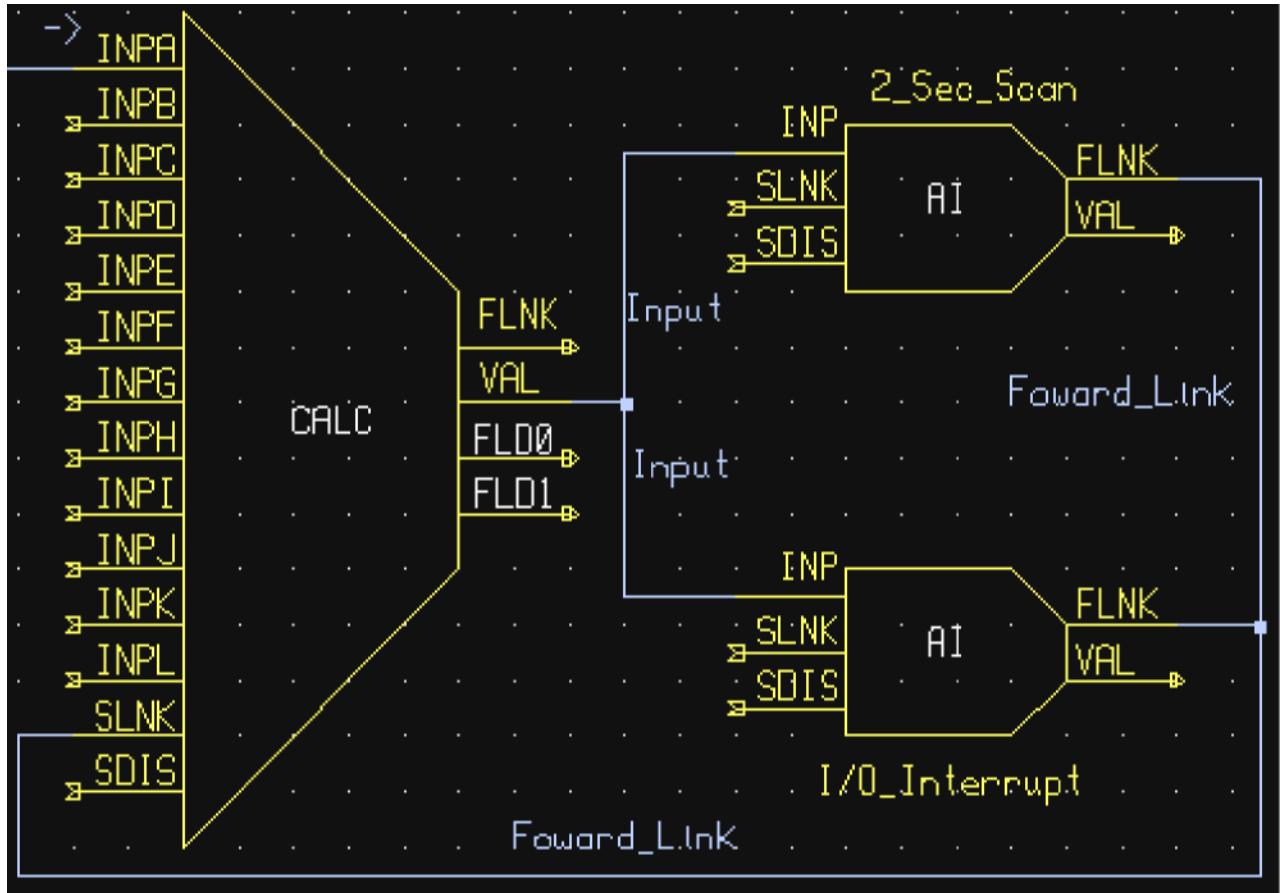
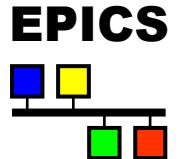
Scan Passive Records

1. Forward Link



- The default scan method for a record is Passive
- A Passive record will not process without some external action
- A Passive record can be processed using a Forward Link from another record. See following example.

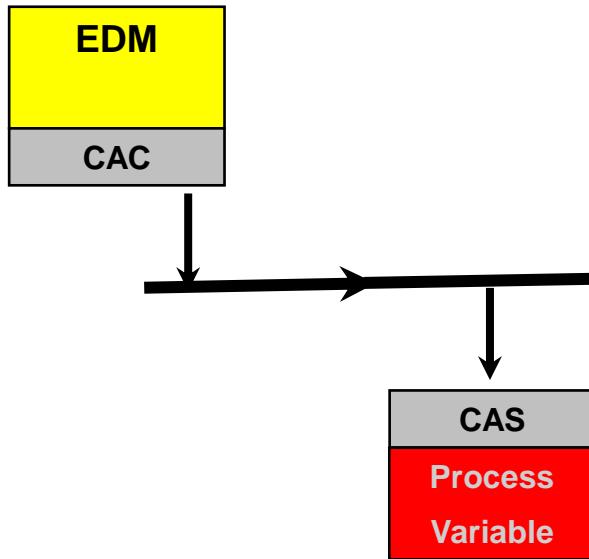
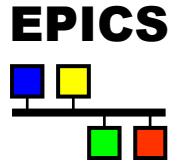
Scan Passive Example: Forward Link



- In this example, the CALC record is *Scan Passive* and is triggered by forward links from two AI records
- The CALC record will be processed every 2 seconds *as well as* when an I/O interrupt occurs

Scan Passive Records

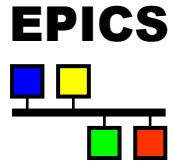
2. Channel Access Put



- It is possible to define a “pp” attribute on any non-link field in a record. This can be set to “TRUE”.
- This is done once in the “dbd” file – not dynamically configured.
- If it is set to “TRUE”, then a channel access “put” to that record field (e.g. VAL field) will cause the record to process.
- This is completely separate to the dynamic attributes defined on a link field.
- A Channel Access put is not visible on a schematic – data source is outside IOC.

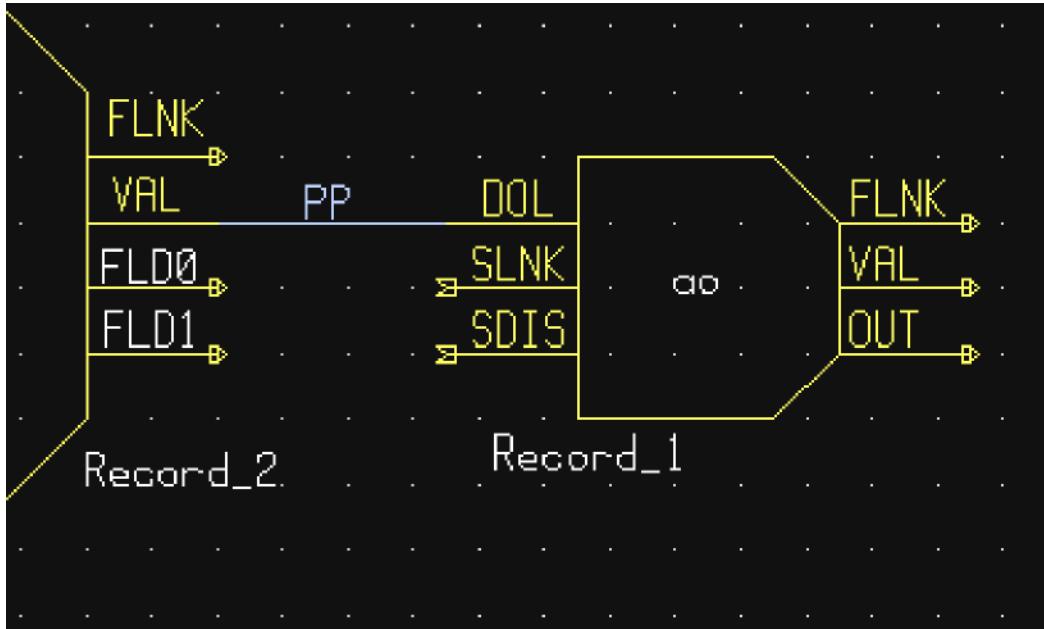
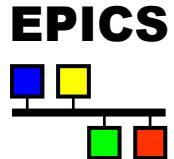
Scan Passive Records

3. Database Link



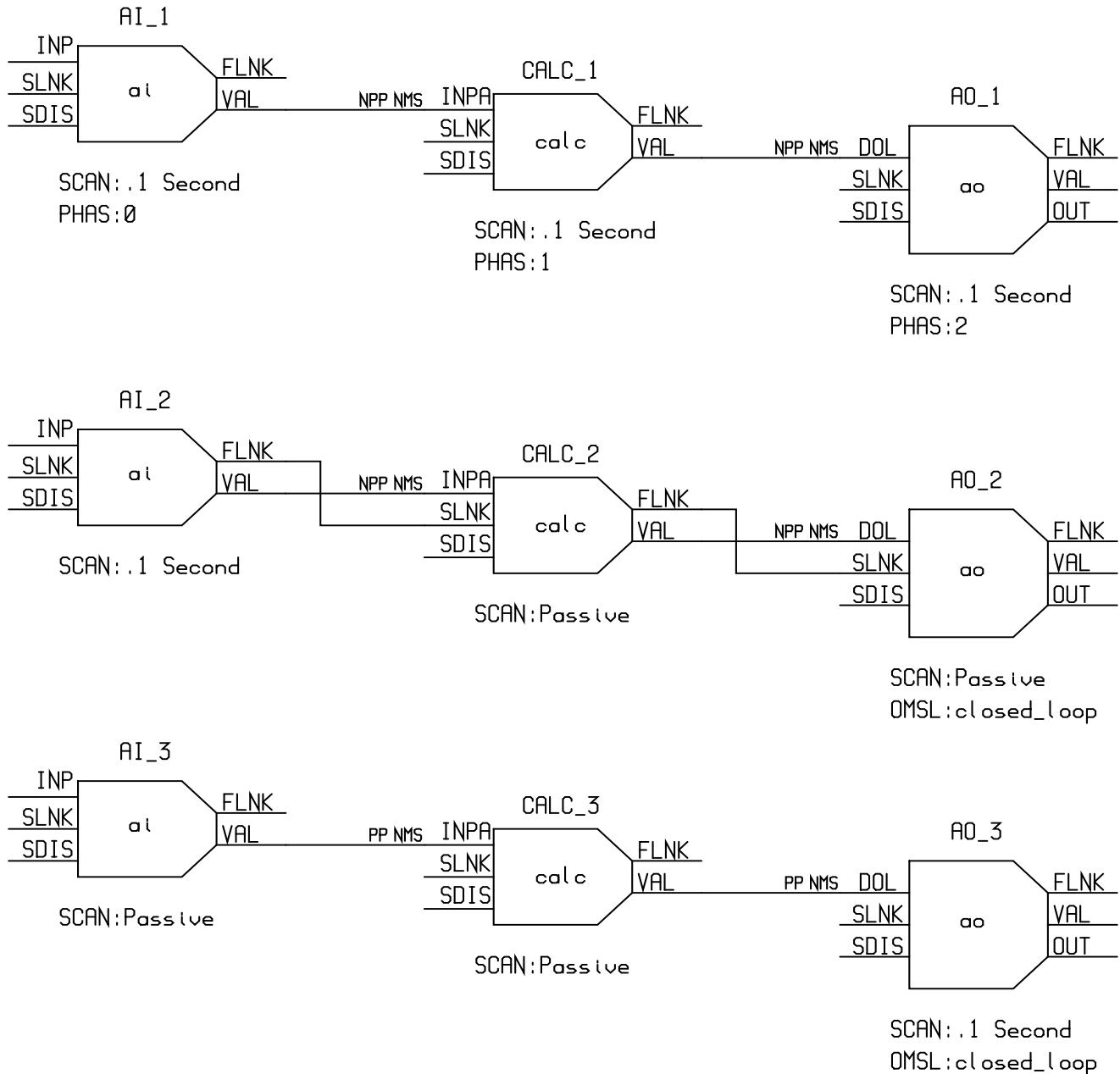
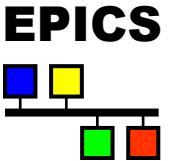
- A Passive record will also be processed when another record reads from *or* writes to a field with the dynamic “PP” attribute set on the database link.
- See following example.

Scan Passive Example: Database Link

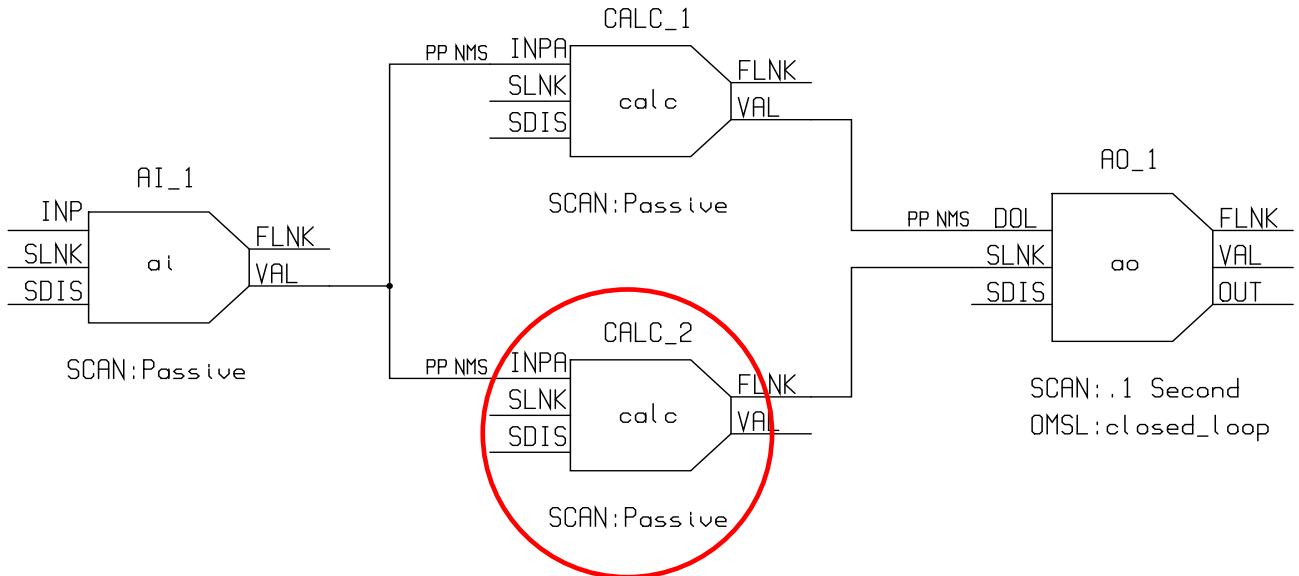
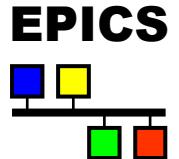


- Both records are *Scan Passive*
- *Record_1* retrieves its output value from *Record_2* via its DOL link
- The DOL link has the **.PP** attribute specified, so when *Record_1* processes it will trigger processing of *Record_2*
- When *Record_2* has finished processing, the value in its VAL field will be retrieved via DOL; *Record_1* will then finish processing

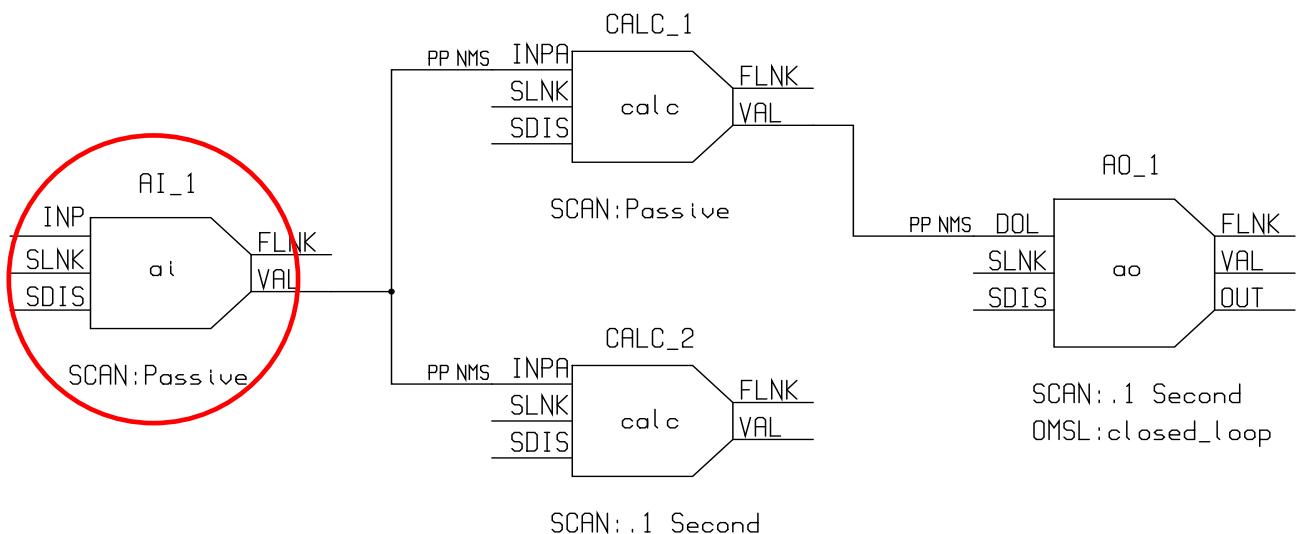
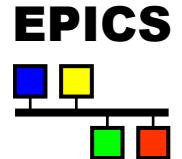
Processing Chains



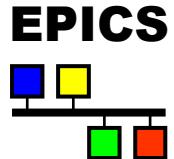
Which record is never processed?



How often is AI_1 Processed?

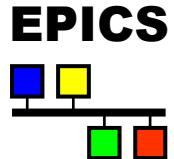


Preventing records from processing



- It is useful to be able to stop an individual record from processing on some condition
- Before record-specific processing is called, a value is read through the SDIS input link into DISA
- If DISA=DISV, the record will not be processed
- A disabled record is alarmed by giving the desired severity in the DISS field
- The FLNK of a disabled record is not triggered

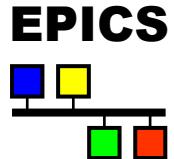
How are records given CPU time?



Several tasks are used:

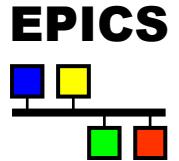
- callback (3 priorities) – I/O Interrupt
- scanEvent – Soft Event
- scanPeriod – Periodic
 - A separate task is used for each scan period
 - Faster scan rates are given higher task priority
- Channel Access tasks use lower priority than record processing
 - If a CPU spends all the time doing I/O and processing, you will be unable to control or monitor the IOC via the network

Lock-sets



- Prevent a record from being processed simultaneously from two scan tasks
- A lock-set is a group of records interconnected by:
 - Output Database links
 - Forward links
 - Input links
 - Arrays (because not an atomic operation)
- Lock-sets are determined automatically by the IOC at start-up
- If they become very big, performance can be affected
- You can split a lock set with Channel Access links, using .CA flag

Alarms



- Every record has the fields

SEVR Alarm Severity

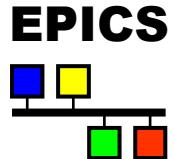
 ↳ NONE, MINOR, MAJOR, INVALID

STAT Alarm Status (reason)

 ↳ READ, WRITE, UDF, HIGH, LOW, STATE, COS,
 CALC, DISABLE, etc.

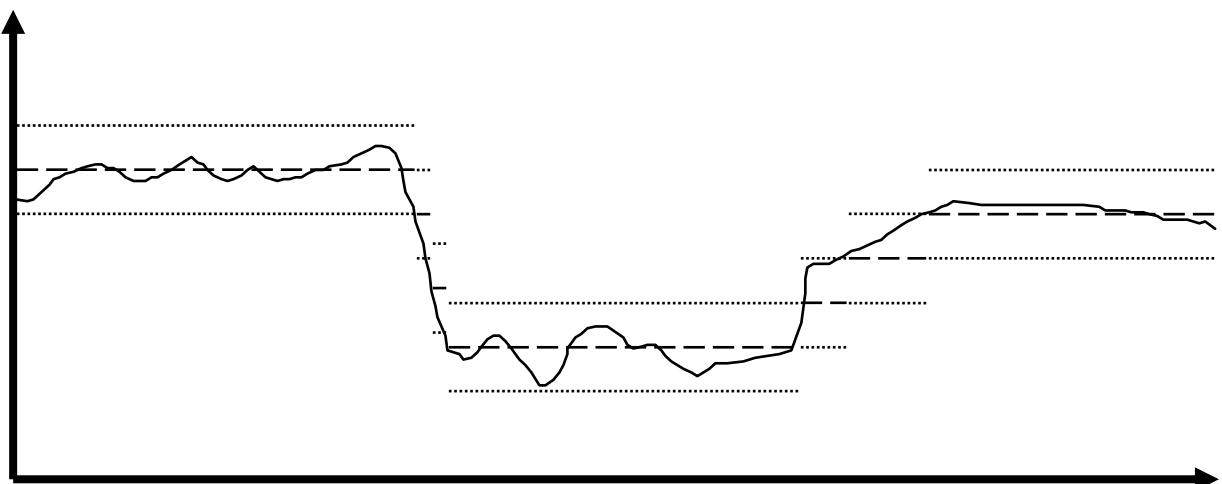
- Most numeric records check VAL against HIHI, HIGH, LOW and LOLO fields after the value has been determined
- The HYST field prevents alarm chattering
- A separate severity can be set for each numeric limit (HHSV, HSV, LSV, LLSV)
- Discrete (binary) records can raise alarms on entering a particular state, or on a change of state (COS)

Change notification: Monitor deadbands

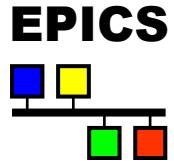


Channel Access notifies clients which are monitoring a numeric record when

- VAL changes by more than the value in field:
 - MDEL Value monitors
 - ADEL Archive monitors
- Record's Alarm Status changes
 - HYST Alarm hysteresis
- Analogue Input record provides smoothing filter to reduce input noise (SMOO)



Breakpoint Tables



- Analogue Input and Output records can do non-linear conversions from/to the hardware data
- Breakpoint tables interpolate from given table
- To use, set the record's LINR field to the name of the breakpoint table you want to use
- Example breakpoint table (in your .dbd file)

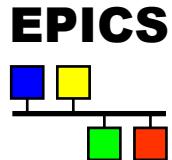
```
breaktable(attenuator1_1) {  
    504      0  
    795      1.25  
    909      2.5  
    1012     3.75  
    ...  
}
```

Simulation

- Input and output record types often allow simulation of hardware interfaces
 - SIMM Simulation mode (YES/NO)
 - SIML Simulation mode link (sets SIMM)
 - SIOL Simulated data value input link
 - SVAL Simulated value
 - SIMS Simulation alarm severity
- Before calling device support, a record reads SIMM through the SIML link
- If SIMM=YES, device support is ignored; record I/O uses the SIOL link instead
- An alarm severity can be set whenever simulating, given by Sims field



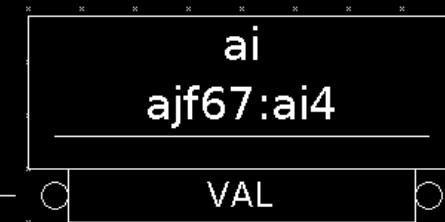
Link Attributes for INPUT links



Turn this database link into a channel access link. This breaks the lock-set



Places monitor on ai4 which causes ai3 to process when ai4 changes

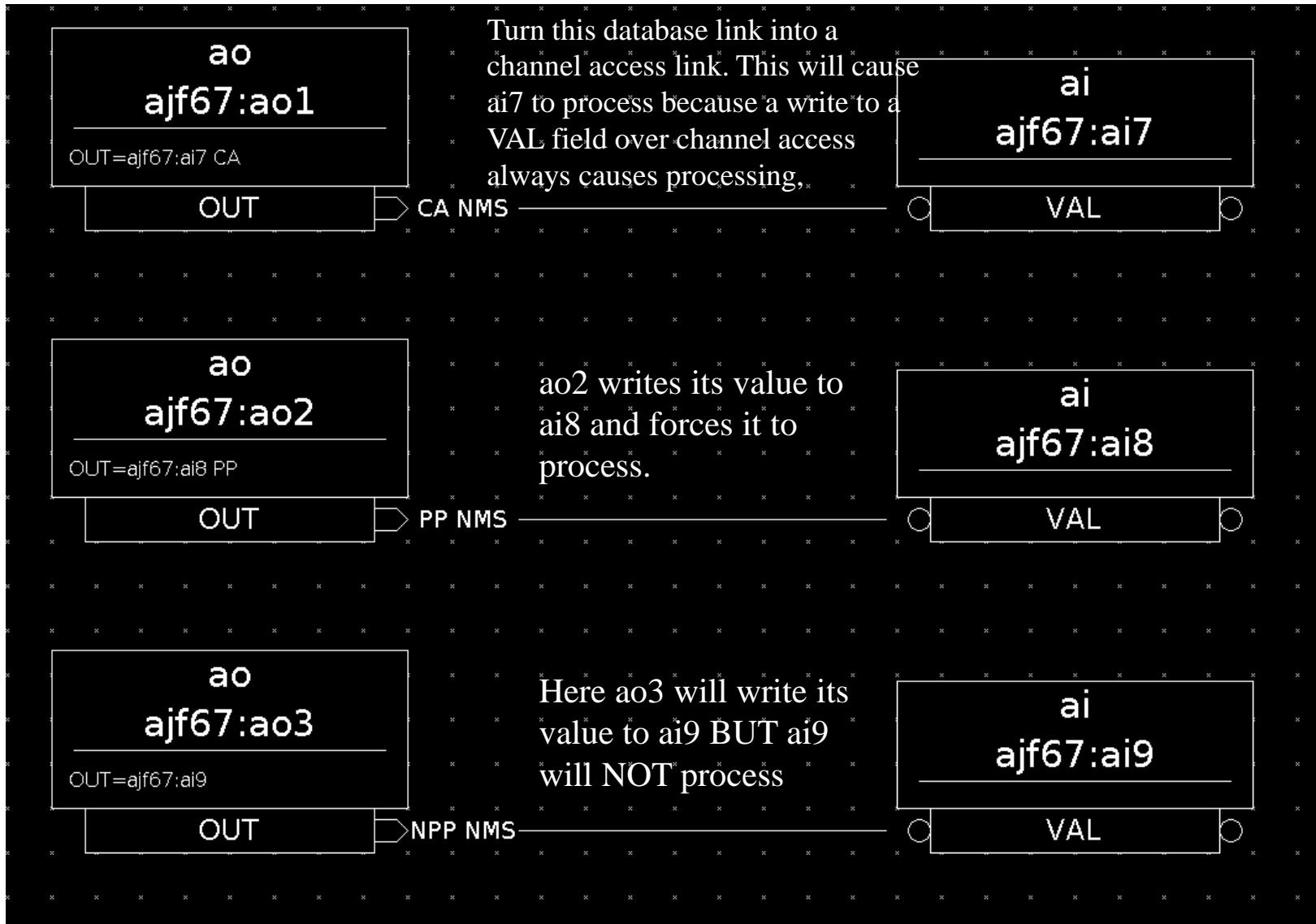
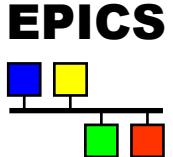


As above, BUT since ai5 is a Passive record, we can use the CPP attribute instead of CP



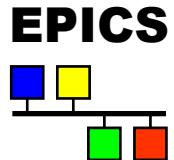


Link Attributes for OUTPUT links





Link Attributes for FORWARD links



The forward link ALWAYS makes the next record in the chain process. No data is passed.



The CA turns this forward link into a channel access link. It will make ai11 process because a write to a VAL field over channel access always causes processing.



Here we are using the forward link to cause a record in a DIFFERENT IOC to process. This link will therefore be a channel access link by default.

EDM

Extensible Display Manager for EPICS

99%: John Sinclair, June 25, 2001

Updated: Kay Kasemir, April 2002

Philip Taylor 2005-2018

Introduction

- EDM is an interactive:
 - GUI builder (for creating screens) *and*
 - Execution engine (for displaying the EPICS database in real-time).
- Also known as a *Display Manager*
- Was maintained by Oak Ridge NL
- Component based, thus extensible by users
- Now being superseded by other EPICS Display Managers, for example CSS/BOY at DLS

Example EDM Operator Screens

The screenshot shows a graph of Value vs Time [s] from -120 to 0. The graph displays three data series: snsiooc4.cpu (red), snsiooc4.memory (white), and snsiooc4.fdescr (yellow). The values are constant at approximately 11.83, 17.84, and 26.00 respectively.

(SNS Linac test)

/export/home/epics/tmp/steiner/edm/CouplingLine.edl
Barney running...

Coupling Line Beamlne Controls

Beam: $^{40}\text{Ar}^{7+}$

New BRho: **2.5368 Tm**
New vs. Now 0.0 %

Store	Rcl	2.5622
Store	Rcl	2.5368
Recall Line		Last 2.5368

Optics: K5t1.data
Magnetic Rigidity 3.3000 Tm

Set: 1 1/2 1/10 % Apply BRho

Do Stuff

Monitor Choice: 1 2 3 4 5 6 Camera 15

Vwr in FC in out Camera 1 off on Camera off on

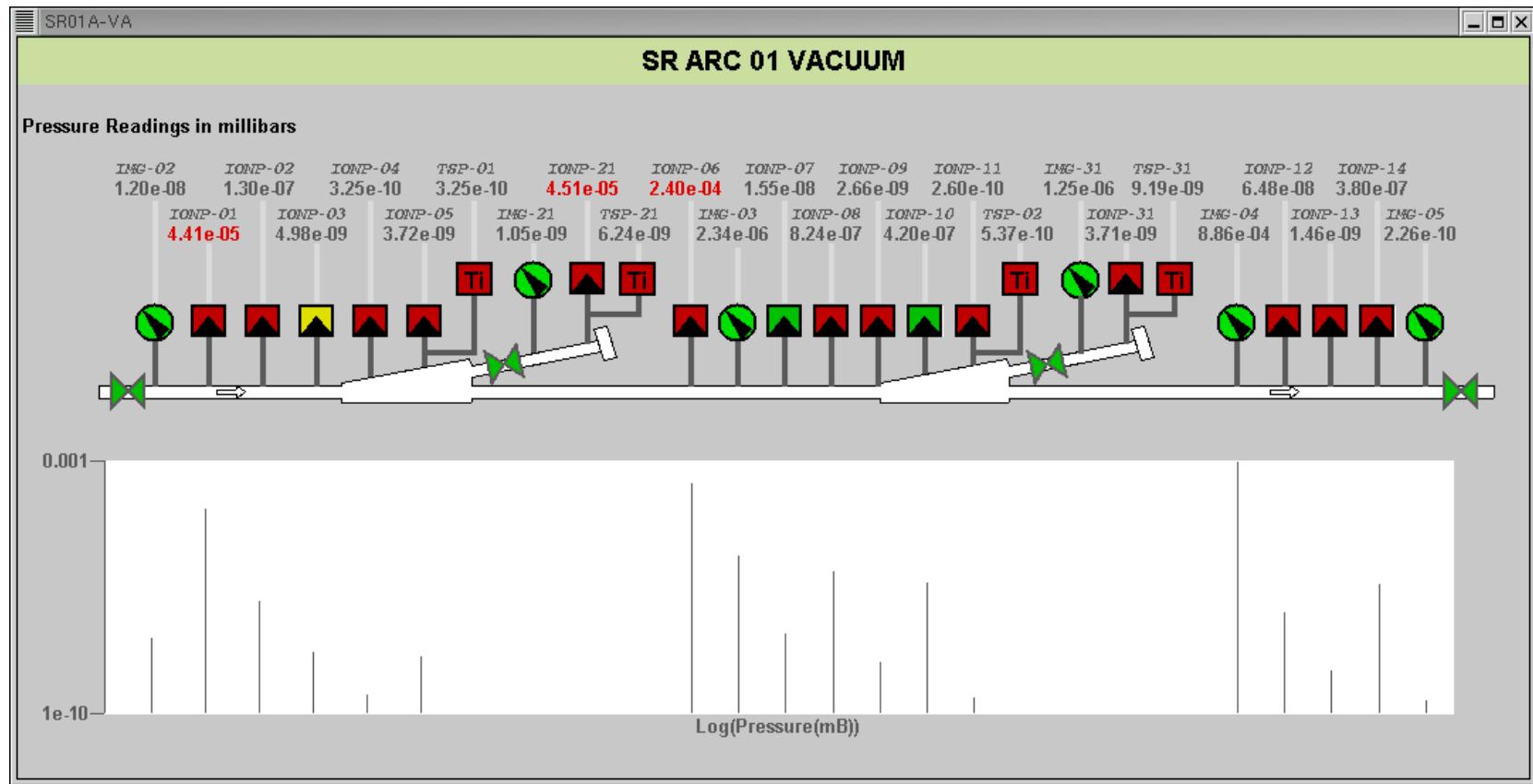
Vwr tg FC tg Camera off on

VFC2 in out SFC2 in out

(Matthias Steiner, Nat'l Superconducting Cyclotron Lab., Michigan State University)

3

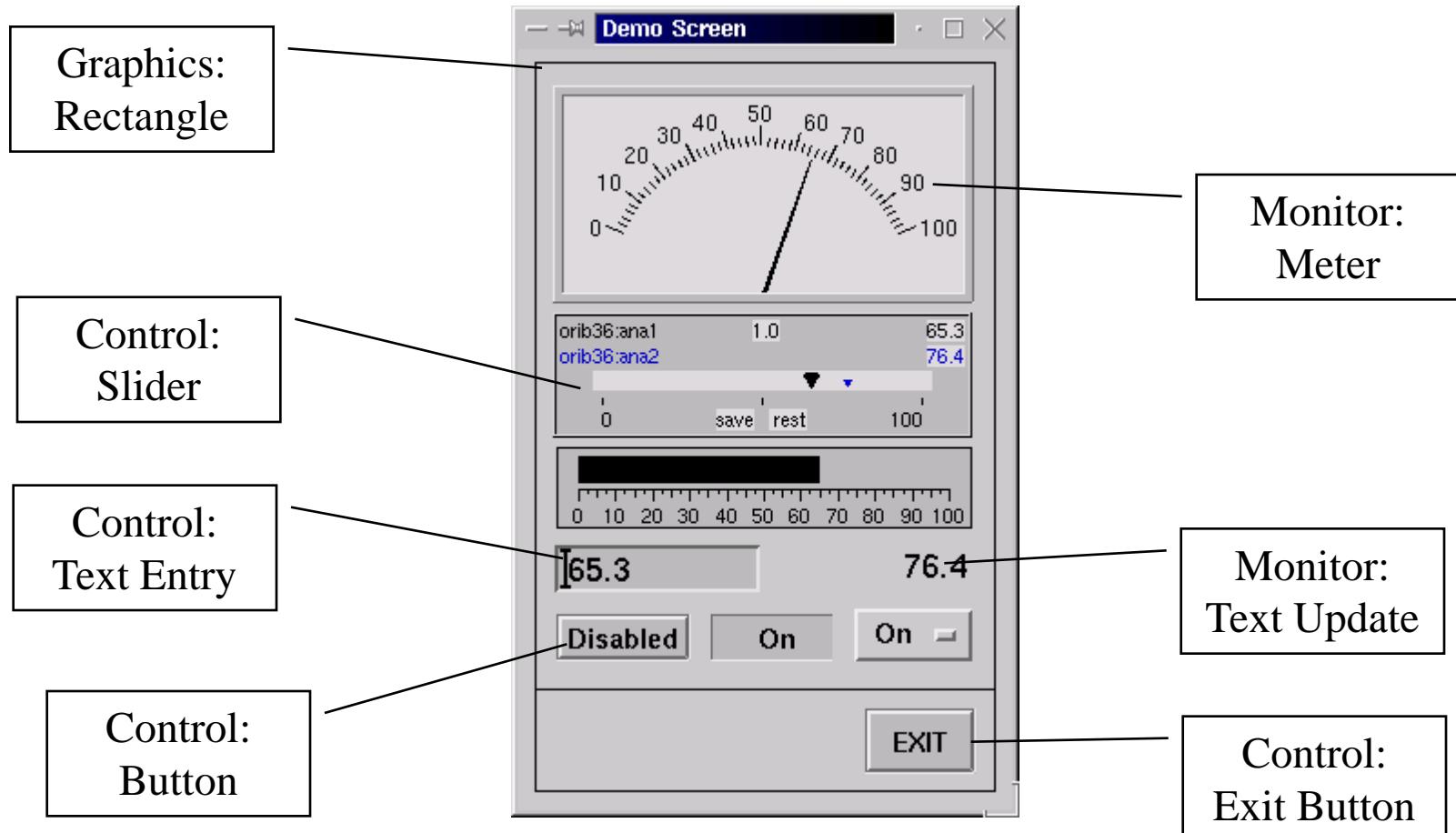
Example DLS Synoptic Display



Example DLS Engineering Display



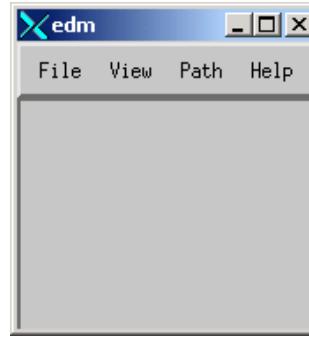
EDM Objects: Examples



Extensible defined as:

- All “objects” are loaded from shared libraries
- EDM administrator can add & remove objects from the list of available objects without recompiling EDM itself
- Objects are versioned; carefully coded objects can be upgraded without impacting existing displays

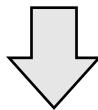
EDM Main Window



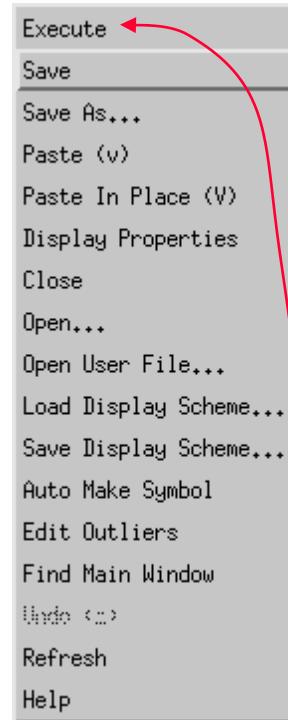
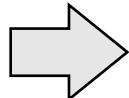
- Change directory to “exerciseApp/opi/edl”
- It is important that “edl” source files are kept in this directory for the exercises
- Startup: only a menu bar, rest of the window is not used:
 - File/New – Create new display
 - File/Open – Open existing display
 - Help – explains many editing features and explains properties of most objects

Display Menu File Operations

With
no objects selected(!)
in a display screen,
click the
middle mouse button
on the display background



This menu pops-up



Save
Save As...
Close
Open...
Open User File...
and:
**Switch between
edit and
execute mode**

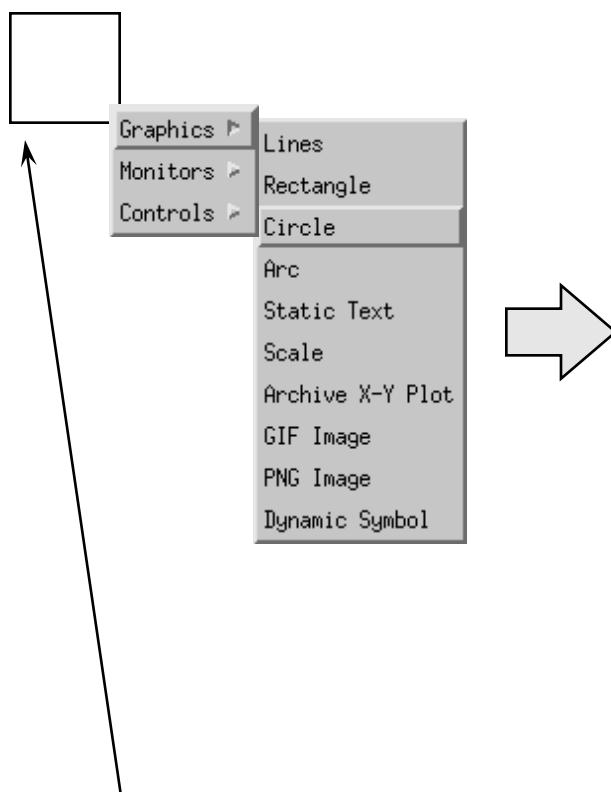
File Operation Notes

- You never need to include the file extension (`xxx.edl`) in a file open or save operation
- “Save As...” to an existing file requires user confirmation

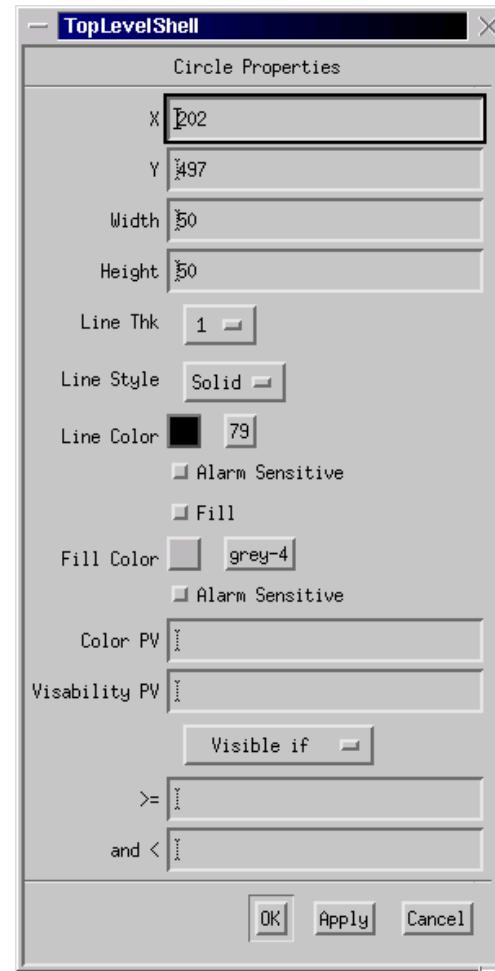
Creating/Editing Displays

- All mouse buttons, many keys and most of the conceivable combinations of shift/ctrl/double-click are used!
- Takes some getting-used-to, but in the end allows for *very efficient editing*.
- Most of the time, no need to use many of these features
- If lost: Press ESC, left-double-click somewhere on the display where there is no object.

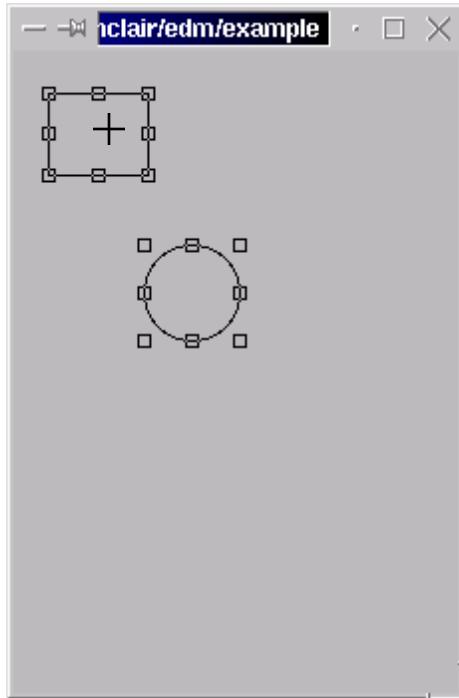
Creating Objects



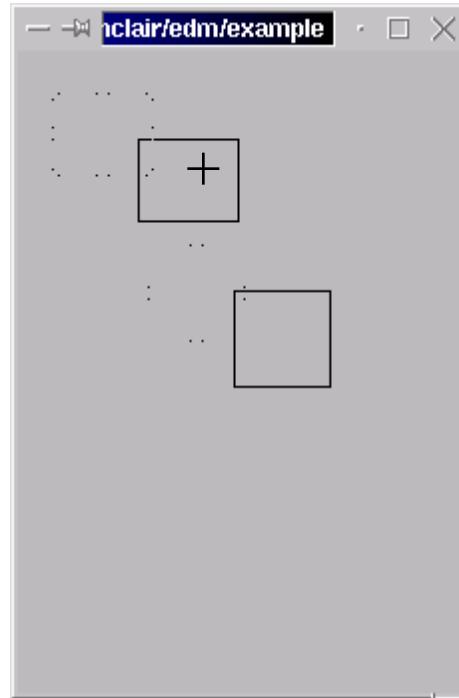
Left mouse button drag to
“rubberband” initial object size



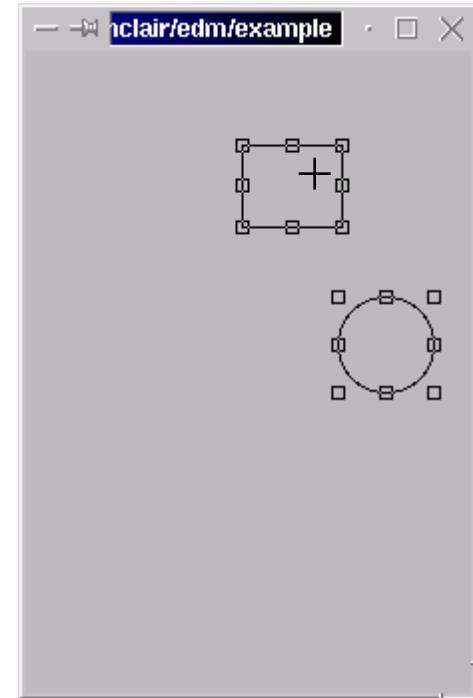
Moving Objects



Place mouse cursor
on interior of one
object

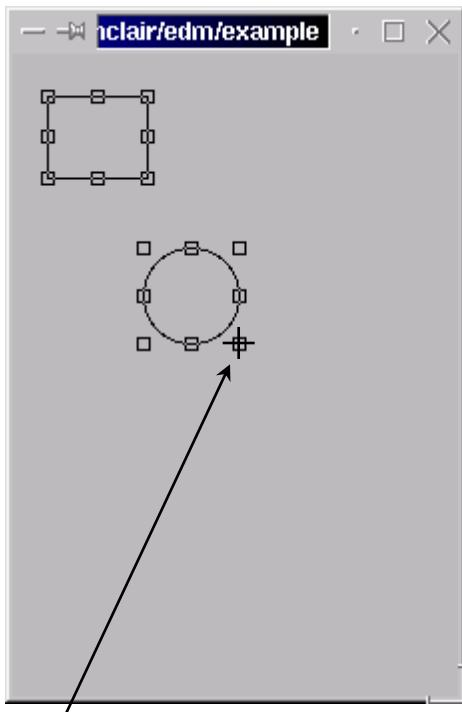


Press left button and
drag objects to new
location

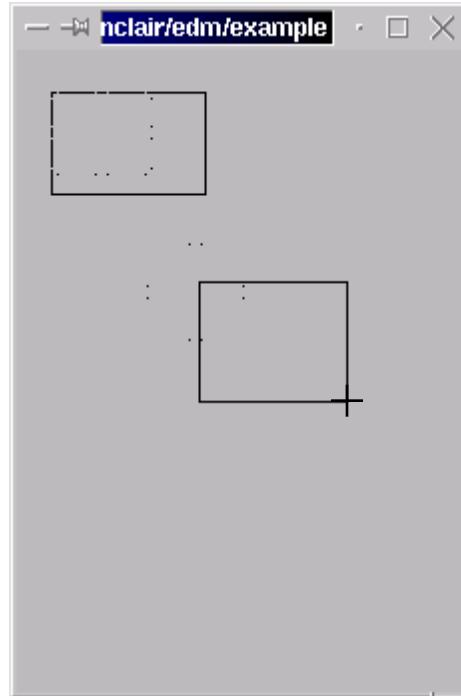


Release mouse
button

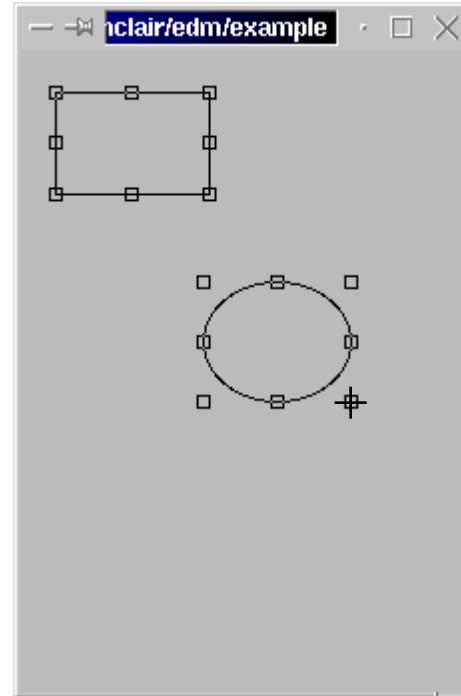
Resizing Objects



Place mouse cursor
on control point
of one object



Press left button and
drag to new size



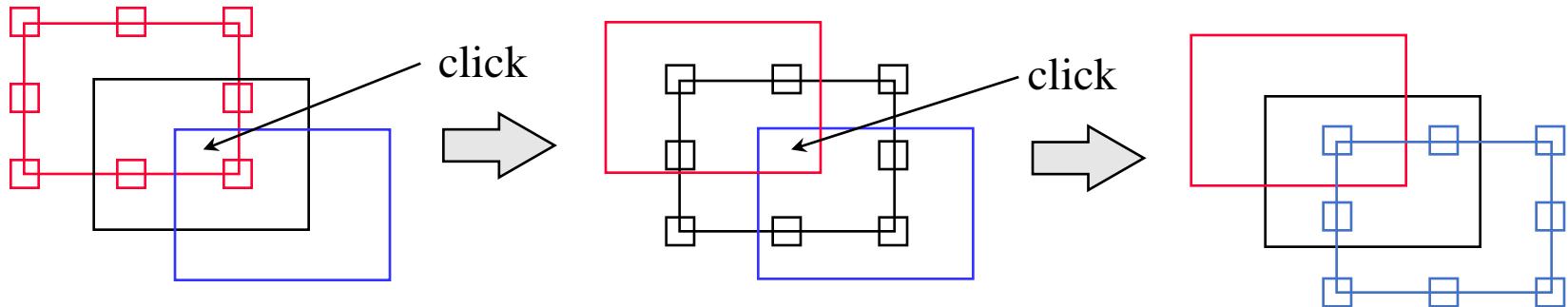
Release mouse
button

Selecting Objects

- Left button click
 - Single exclusive select: object is selected, currently selected objects are deselected
- Shift-left button click
 - Single inclusive select: object is added or removed from the current group of selected objects

Selecting Objects (cont)

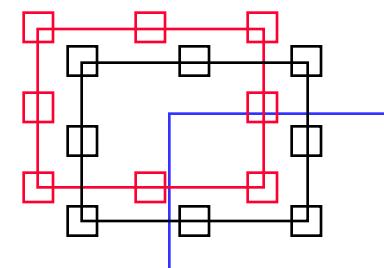
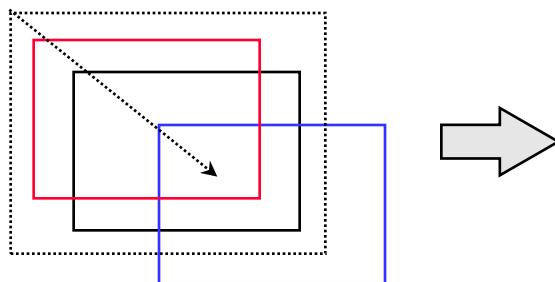
- Control-left button click
 - If only one object is currently selected then selection *cycles* among overlapping objects



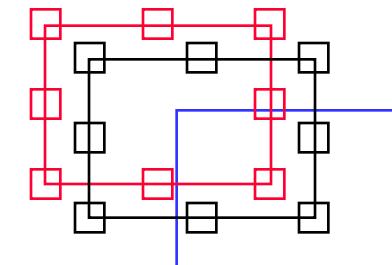
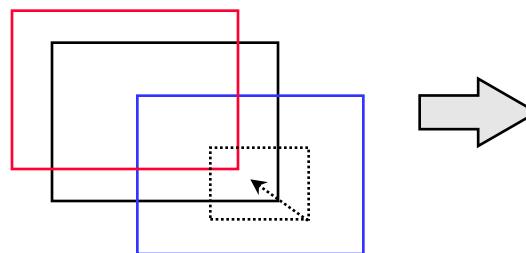
This convenient idea was adopted from AutoCad...

Selecting Objects (cont)

- **Middle** button drag - objects are added or removed from the current selection group



Top-left to bottom-right:
Select enclosed **objects**

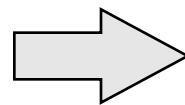
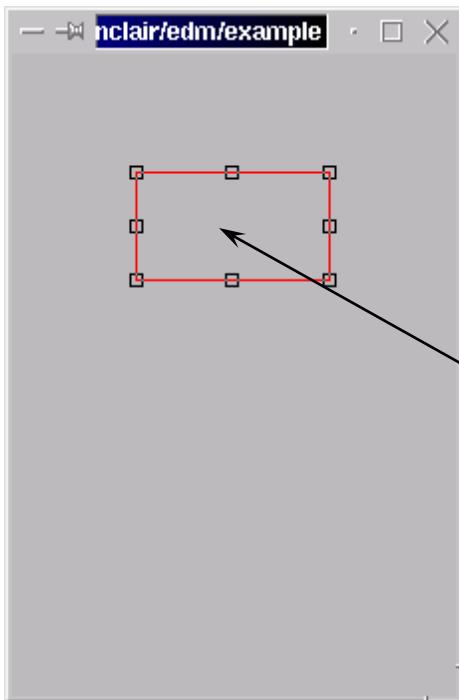


Bottom-right to top-left:
Select enclosed **corners**

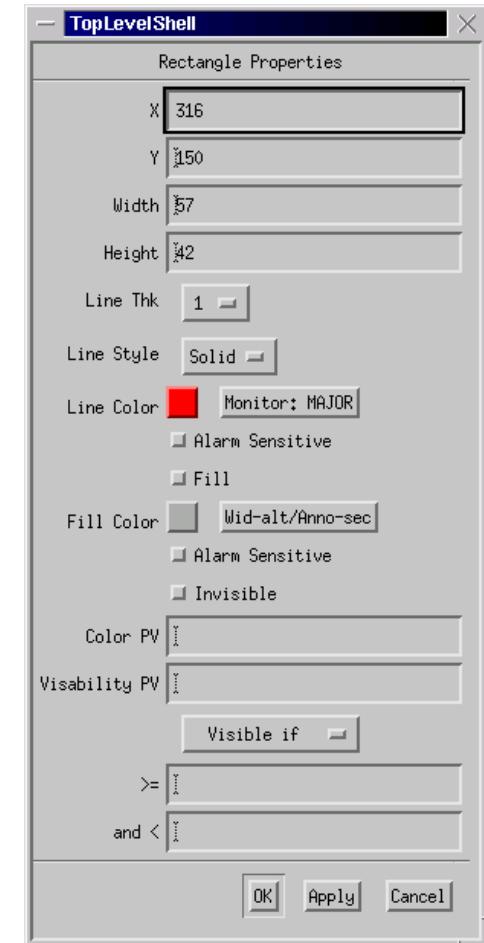
Again:

- **Left** button rubberband: Create new object
- **Middle** button rubberband: Select objects

Editing Objects: Property Dialog



left click on
selected object



Note:
Property dialog varies
with Object type...

Draw/Move/Resize Notes

- Fine control may be achieved on moves and resizes by using *keyboard arrow keys* (mouse button release or click ends op)
- Control key forces move (prevents resize)
(Useful for tiny objects where you cannot click “inside” w/o hitting the resize handles)
- Shortcuts to options in the Display Properties
 - M/m key turns ON/off orthogonal move
 - L/l key turns ON/off orthogonal line draw
 - G/g key turns ON/off grid
 - S/s key turns ON/off snap-to-grid

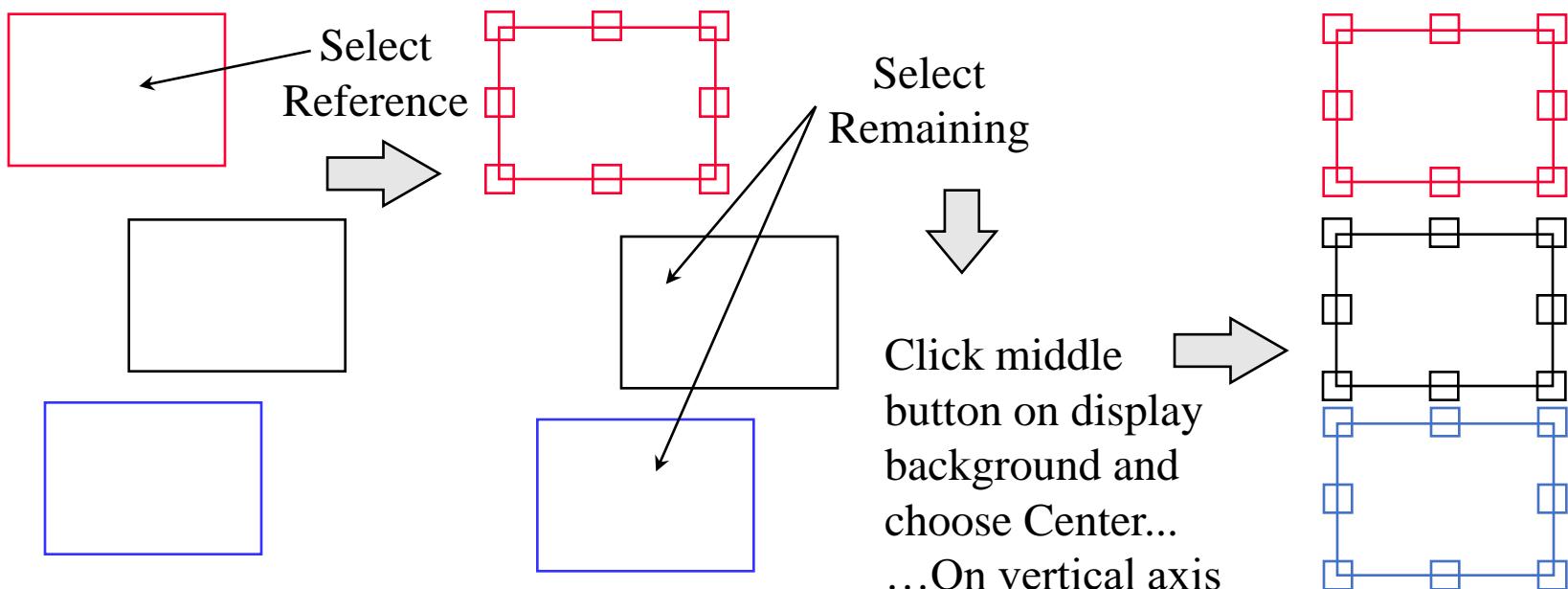
Alignment Operations

- Reference Independent
 - Align left, right, top, bottom
 - Distribute: vert axis, horiz axis
 - Distribute Midpoint: vert axis, horiz axis
- Reference Dependent
 - Center: horizontal, vertical, both
 - Size: width, height, both

Reference Dependent Operations

- First object selected is used as reference
- If no reference object is specified, an appropriate object is chosen (topmost, leftmost, etc.)

Example Align Operation



Misc. Operations

- Raise, Lower
- Copy, Cut, Paste (key shortcuts: c, x, v)
- Group, Ungroup
 - *Can control visibility of a group of objects*
- Flip H & V
- Rotate CW & CCW
- Group Edit
- Undo

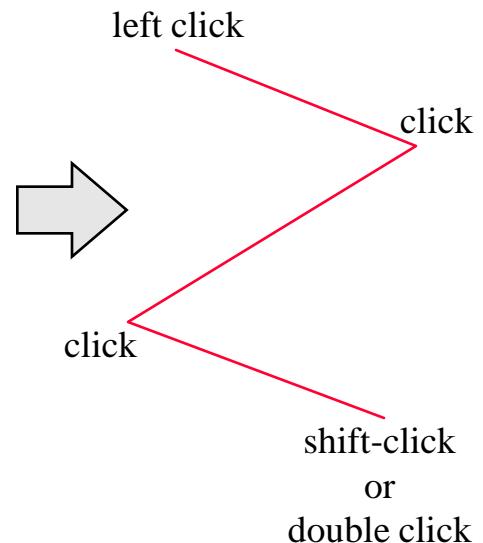
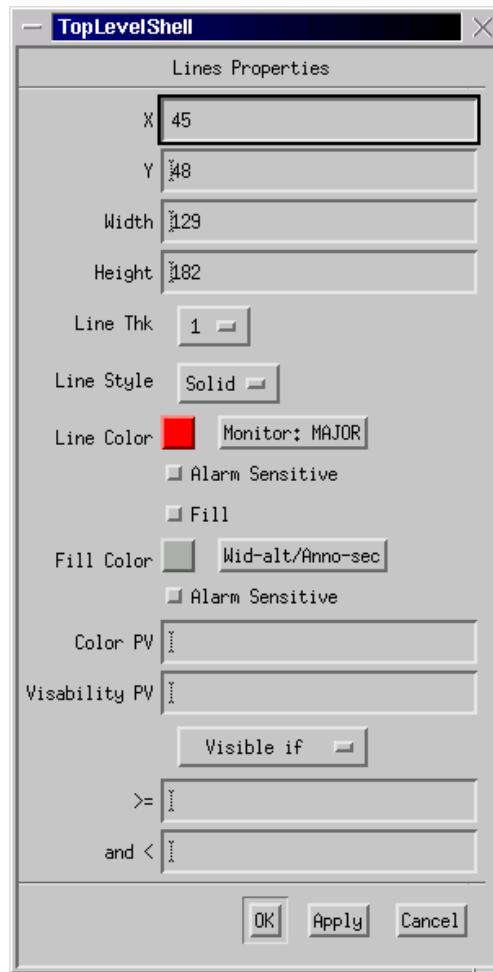
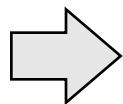
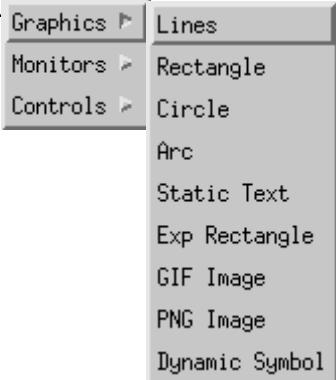
Editing Notes

- Clicking on one of a *group* of selected objects brings up the property box for each object, one-by-one, as the OK button is pressed.
- To minimize mouse movement, instead of clicking OK, Apply, or Cancel, you may *double-click* the left, middle, or right button respectively.

Undo

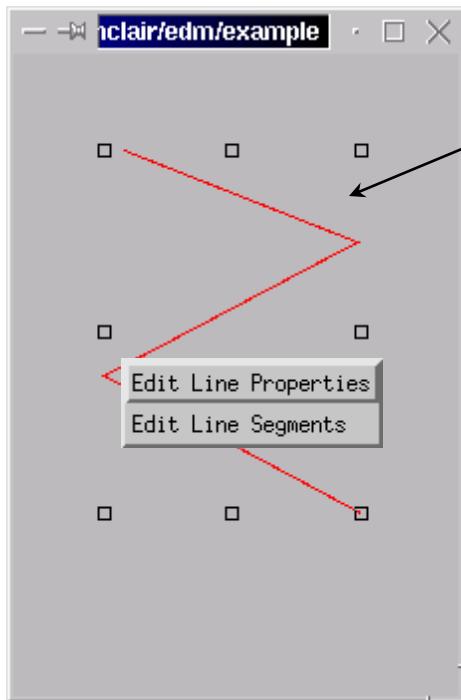
- Most useful for move, resize, & alignment operations
- Current limitations:
 - Cannot undo edit operations
 - Cut, Group, and Ungroup : Flush undo stack

Creating Lines

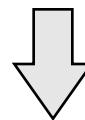


Left mouse button drag

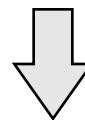
Editing Line Properties



left click on
selected object

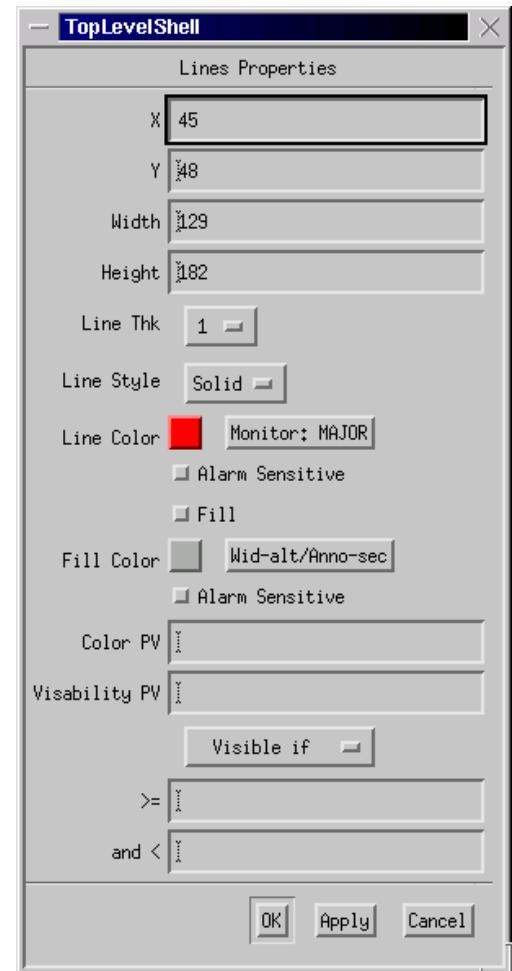


Menu Appears →



Choose Edit
Line Properties

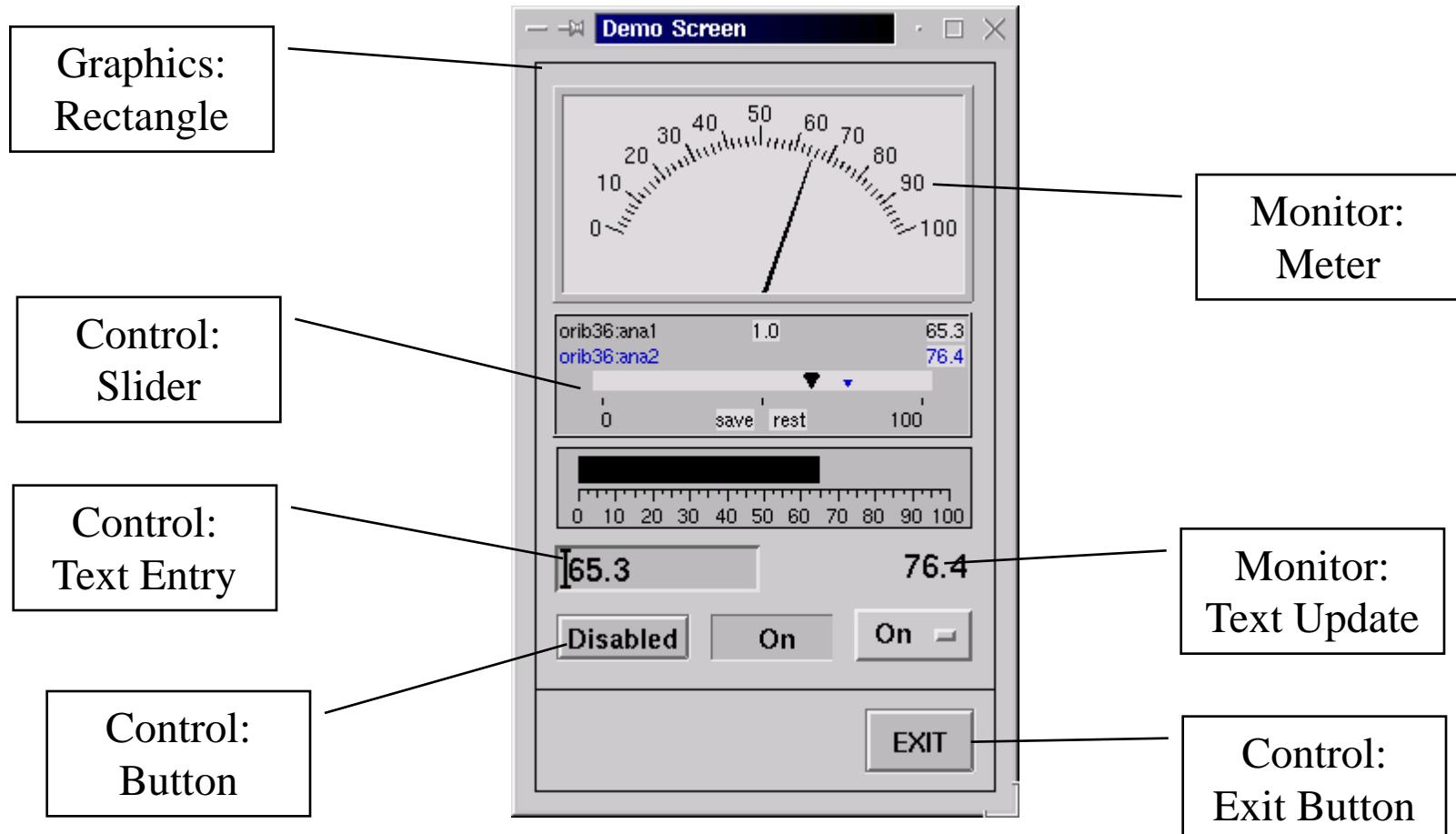
Can also Edit Line Segments



Group Edit

- Change visual attributes of all selected objects
- Change PV names for all selected objects

EDM Objects



Object Categories

- **Graphics**

Do not require a process variable

- Lines, rectangle, circle, arc, text, gif, png, dynamic symbol, embedded window

- **Monitors**

Display current value of process variables

- Meter, bar, message box, symbol, text update, X-Y graph, strip chart, byte, table

- **Controls**

Modify value of process variables, change displays

- Text, slider, button, menu button, message button, up-down button, related display, shell command, ...

Online Help

The screenshot illustrates the online help system of a software application, likely a simulation or control system interface.

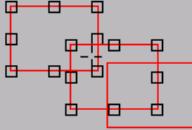
Left Panel (Help Index):

- Objects
- Graphics
- Lines
- R
- All
- St
- C
- G
- P
- Dyna
- Help
- Mouse Op
- File Oper
- Creating C
- Selecting C
- Editing O
- Line Objects
- Aligning Objects
- Objects
- Available Symbols

Middle Panel (Help Topics):

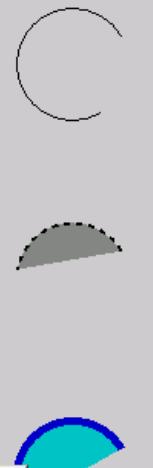
- Help - Creating Lines**
 - 1) Left B drag box and release
 - 2) Select Graphics --> Lines from the menu
 - 3) Select options from property box and click the OK
 - 4) Create all node points
 - o Left B click adds a node point to the line
 - o Shift middle B click deletes the last node point
 - o Middle B drag moves node points
 - 5) Terminate operation
 - o Shift Left B click
 - or
 - o Left B double-click

Help - Multiple inclusive select
Middle B drag from top-left to bottom-right inclusively selects all enclosed objects.
Middle B drag from bottom-right to top-left inclusively selects all objects for which at least one corner falls inside the select box.
This operation adds unselected objects to the select group and removes those already selected.

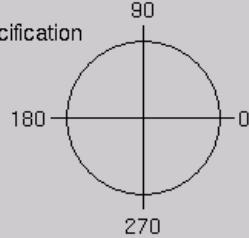


Right Panel (Arc Properties):

Arc



- Line Style: Solid
Line Thk: 1
No Fill
Start Angle: 30
Total Angle: 270
- Line Style: Dash
Line Thk: 2
Fill
Fill Mode: Chord
Start Angle: 30
Total Angle: 160
- Line Style: Solid
Line Thk: 5
Fill
Fill Mode: Pie
Start Angle: 30
Total Angle: 160



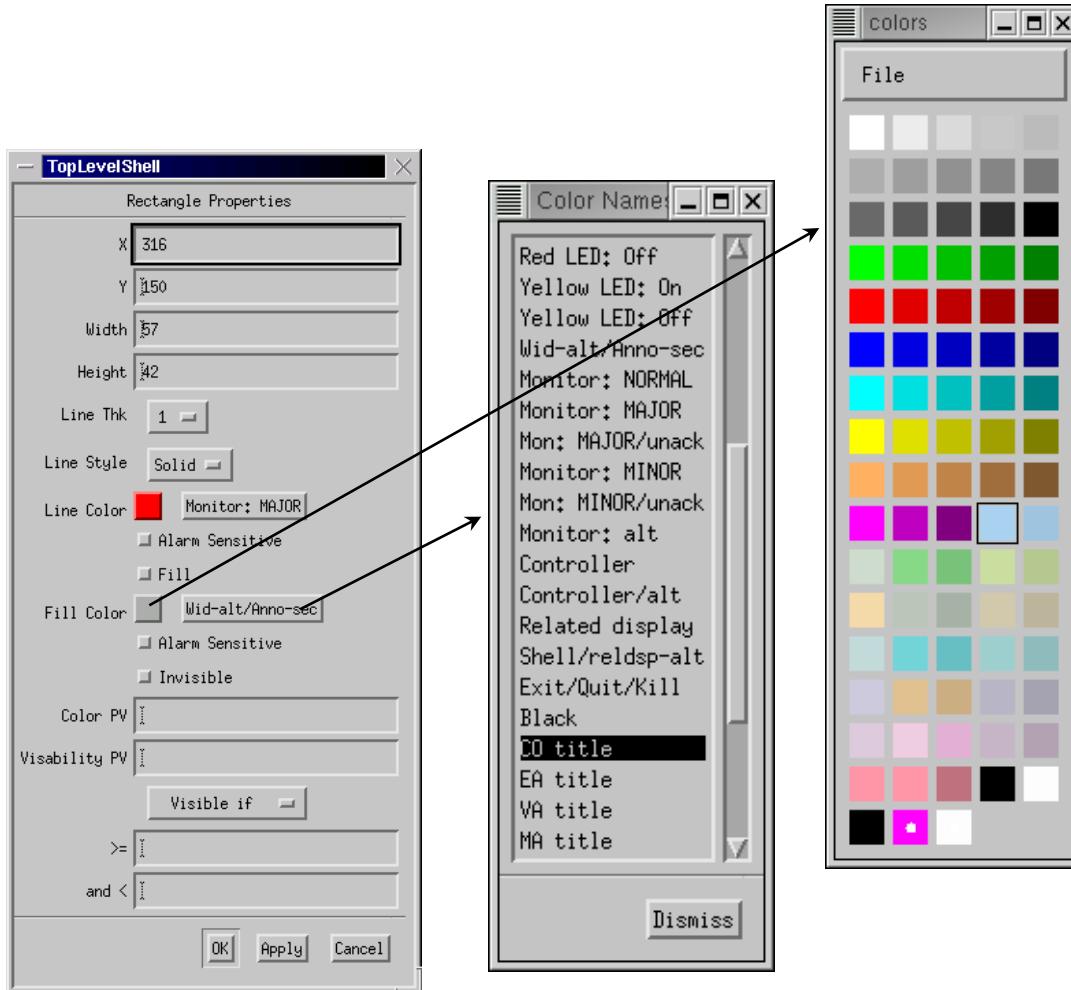
Color PV is used with dynamic colors and alarm sensitivity. If both are present, alarm colors have precedence.

Visibility may be achieved through visibility PV and embedded module or with invisible color.

EPICS Process Variables

- Many EDM objects use EPICS PVs to:
 - show the PV value (Monitors)
 - control the PV value (Controls)
 - change color or visibility based on the PV (all types)
- Often, the same EDM GUI will be re-used across different subsystems
- Use macros in PV names and provide the macro value when starting EDM
 - e.g. \$(P):X:SIZE.RBV

Specifying Color



- Color may be specified visually or by name
- Online help explains the current color file format
- The color palette dialog shows names as “tooltips”
- *Decoration or Meaning?*
Example:
The same shade of red might be available as both “red” and “Monitor: MAJOR”.
Pick the one that fits the desired purpose.

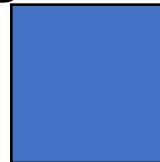
(Diamond EDM Colours and Colour Rules)

Color - Static and Dynamic

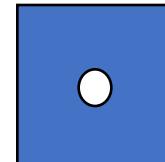
- Colors may be specified for various object attributes e.g. line, fill
- When selecting “alarm sensitive”, the color will change based on the PV alarm severity
- Some color entries are dynamic and are associated with a color rule

Color - Static and Dynamic

- In execute mode, dynamic colors change as a function of the color rule operating on the current value of an associated PV
- Dynamic colors are differentiated from static colors in the following manner:



Static



Dynamic

- For a definition of the color, refer to the colors.list file in the EDM config dir and the online help

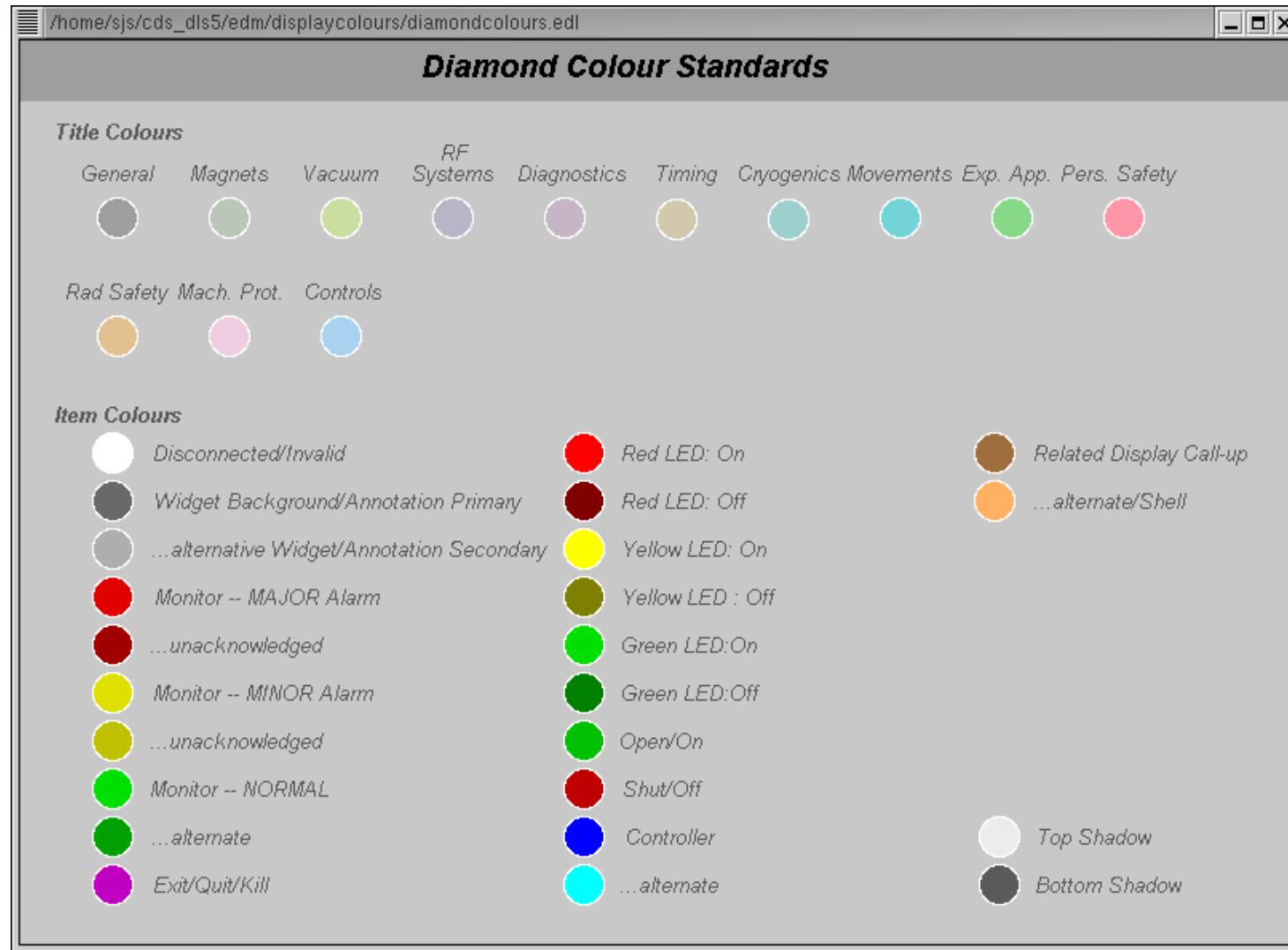
Color Rules

- Color Rules are defined in the edm color.list file. The following is an example of a rule:

```
rule Red-or-Blue
{
    <5          : red
    >=5         : blue
}
```

- This color will be “red” or “blue” depending on the value of the PV.
- Some objects provide a separate “Color PV” that can be used instead of the “main” PV for rule evaluation.

DLS Standard Colour Usage



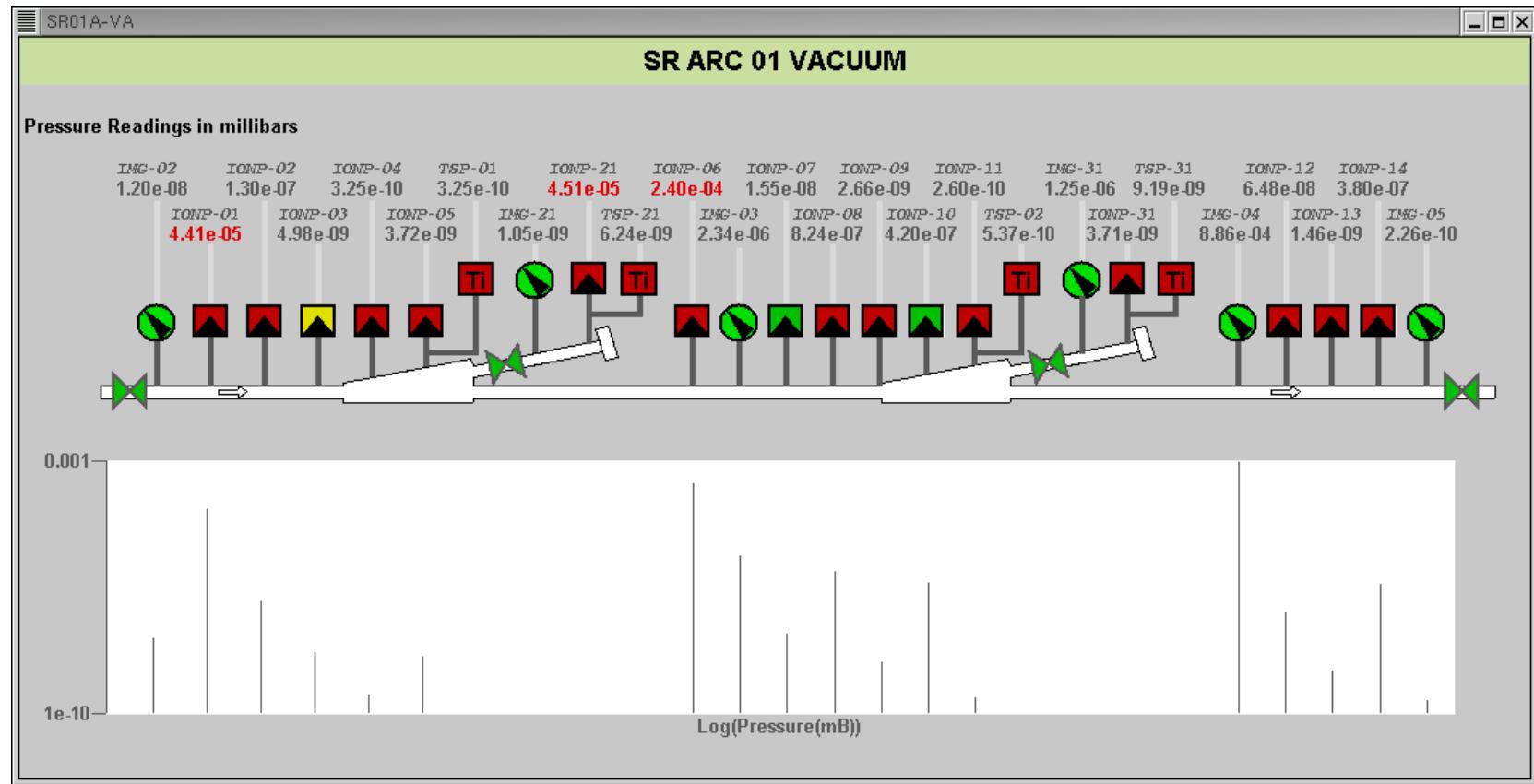
A Word of Warning!

- Never use a white background!
 - Disconnected PVs are displayed in white and will become invisible!
- Instead:
 - Middle click on the canvas
 - Select ‘Display properties’
 - Set ‘Display bg’ to *canvas* (grey)

Symbols

- EDM implements a primitive symbol facility
- Symbols are multi-state objects where each state maps to a value range of an associated EPICS PV
- 64 states max, color and size may be changed per symbol instance if so desired

Example DLS Synoptic Display



Other Display Objects

- Shell Command
 - Used to execute other programs from EDM
- Related Display
 - Used to open up another EDM display

EDM Macro Expansion

- Macro symbol values can be defined in
 - Command line
 - Related Display parameter
 - Multiplexor Object
- At run-time, symbol is substituted with the provided value:

e.g. command line option “-m ‘user=xyz4521’
at run-time, \$(user) → xyz4521

Program Execution - Command Line Options

- Define macro replacement
 - m “var1=value1,var2=value2,...”
(referenced as \$(var1) and \$(var2))
- Execute mode
 - x (-noedit)
- Typical for operations:
edm -x -noedit -m “var1=1,var2=2” displayFile
- In “exerciseApp/opi/edl”, there is a file: “startgui1” which we will be using to run the GUI for exercise 1. It contains the necessary macro substitutions
- In the exercises, we will need to create “startgui<n>” using “startgui1” as an example

Visual DCT

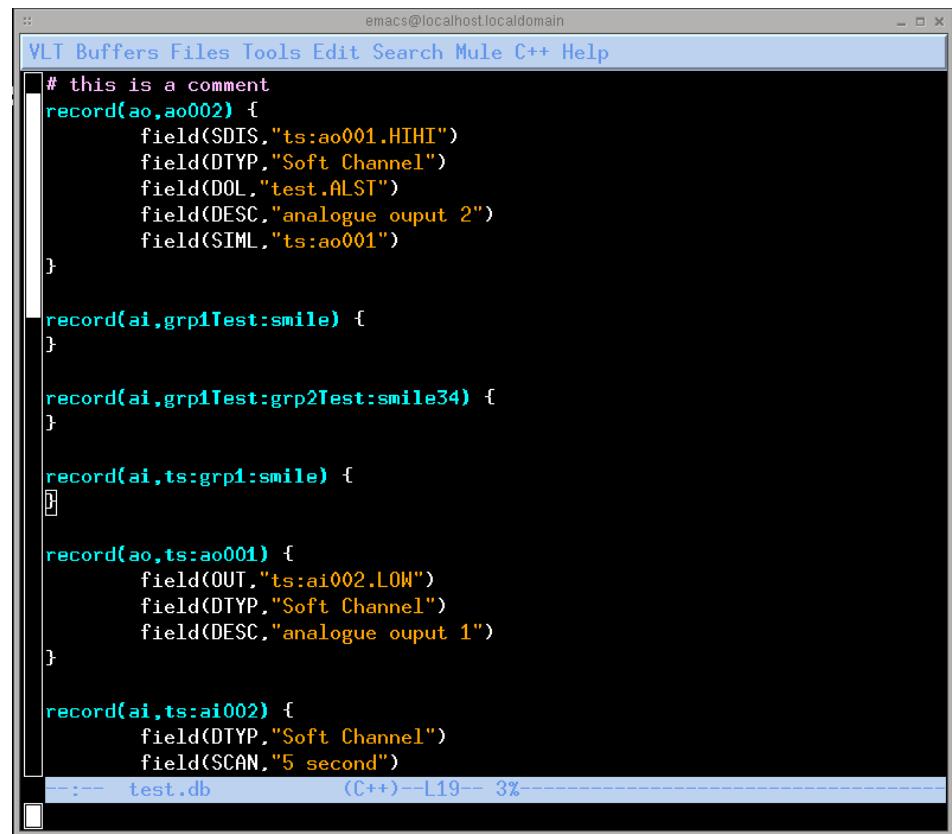
Visual Database Configuration Tool

Original: Matej Sekoranja

Additional material by Nicholas Di Monte (APS) and Philip Taylor

Creating a database

- A database is specified by a text file
- The text file declares the records in the database, specifying their fields and types
- We could create this using a text editor
- However, the declarative nature of a database lends it self to using a “visual editor”
- This is especially useful when you are learning



The screenshot shows a terminal window titled "VLT" with the command "emacs@localhost.localdomain". The file "test.db" is open in the editor. The code defines several records:

```
# this is a comment
record(ao,ao002) {
    field(SDIS,"ts:ao001.HIHI")
    field(DTYP,"Soft Channel")
    field(DOL,"test.ALST")
    field(DESC,"analogue ouput 2")
    field(SIML,"ts:ao001")
}

record(ai.grp1Test:smile) {}

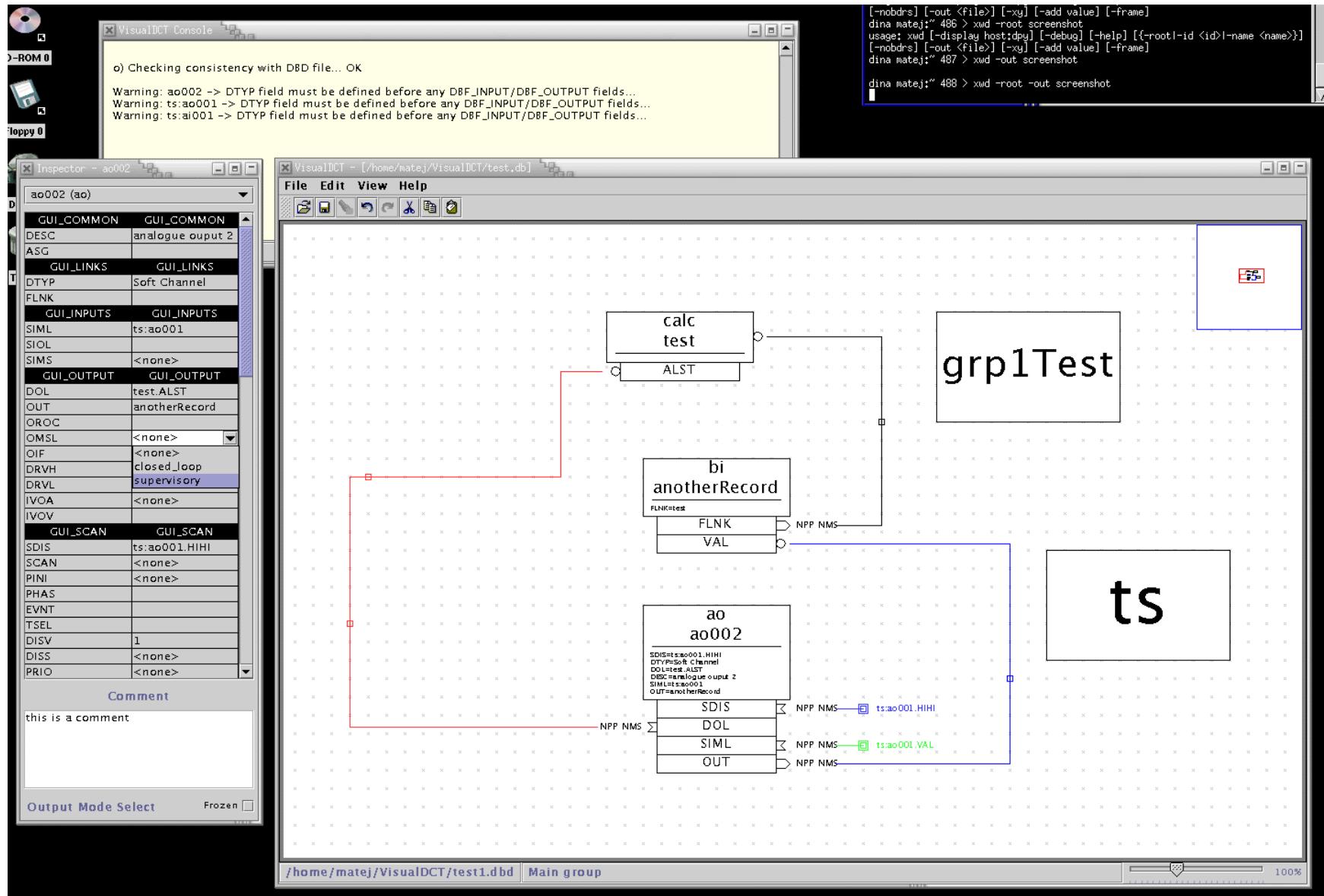
record(ai.grp1Test:grp2Test:smile34) {}

record(ai.ts:grp1:smile) {}

record(ao,ts:ao001) {
    field(OUT,"ts:ai002.LOW")
    field(DTYP,"Soft Channel")
    field(DESC,"analogue ouput 1")
}

record(ai,ts:ai002) {
    field(DTYP,"Soft Channel")
    field(SCAN,"5 second")
}
--:-- test.db (C++)--L19-- 3%
```

A visual editor



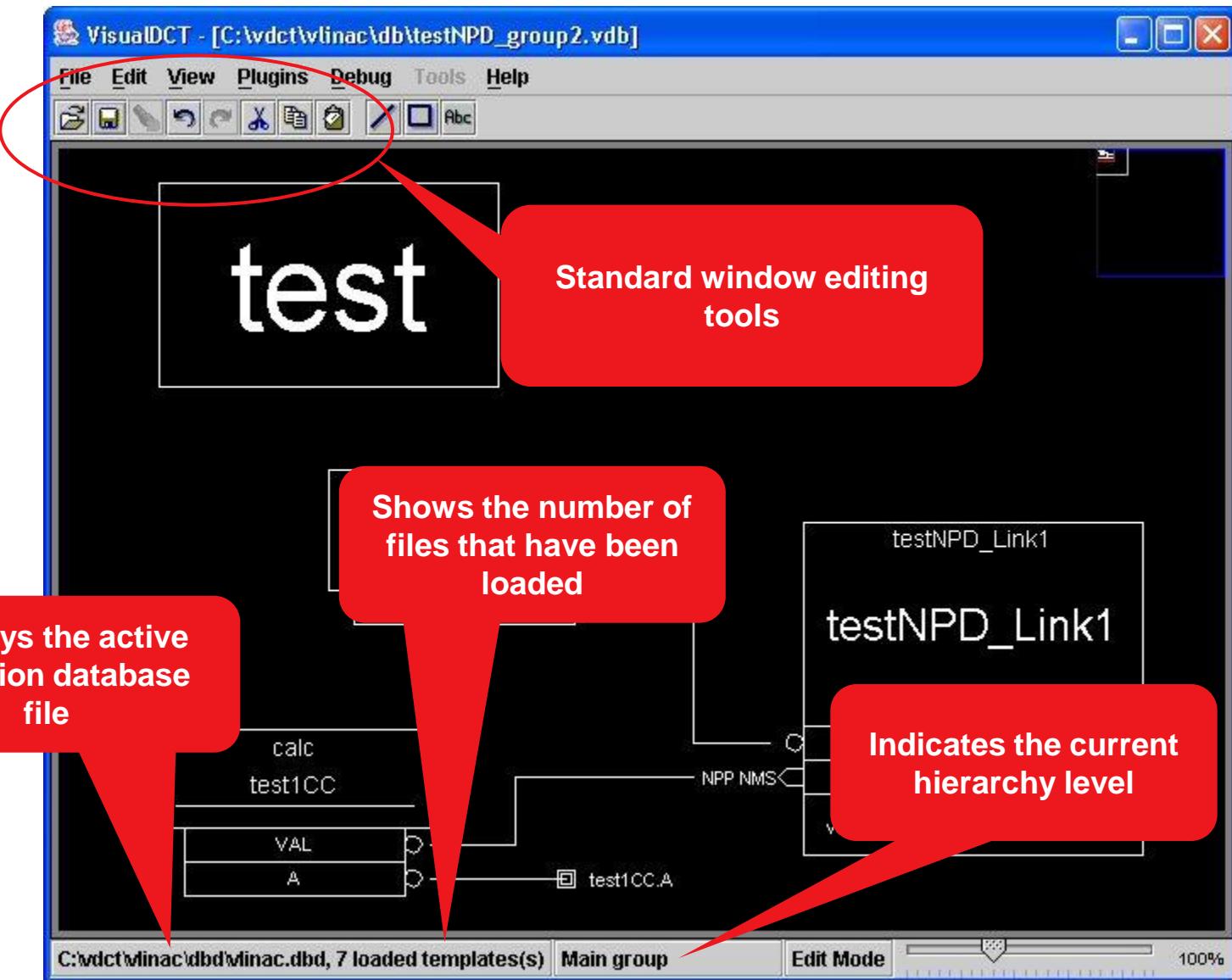
Visual DCT features

- Open source software. Free downloads with no licensing restrictions.
- Developed by Cosylab, funded by a number of institutions (SLS, DLS, APS, ORNL).
- System independent: written in Java
 - Runs on any platform with Java Runtime Environment 2
- Reads DBD file directly
 - Menus available using DBD file definitions
- Outputs DB file directly

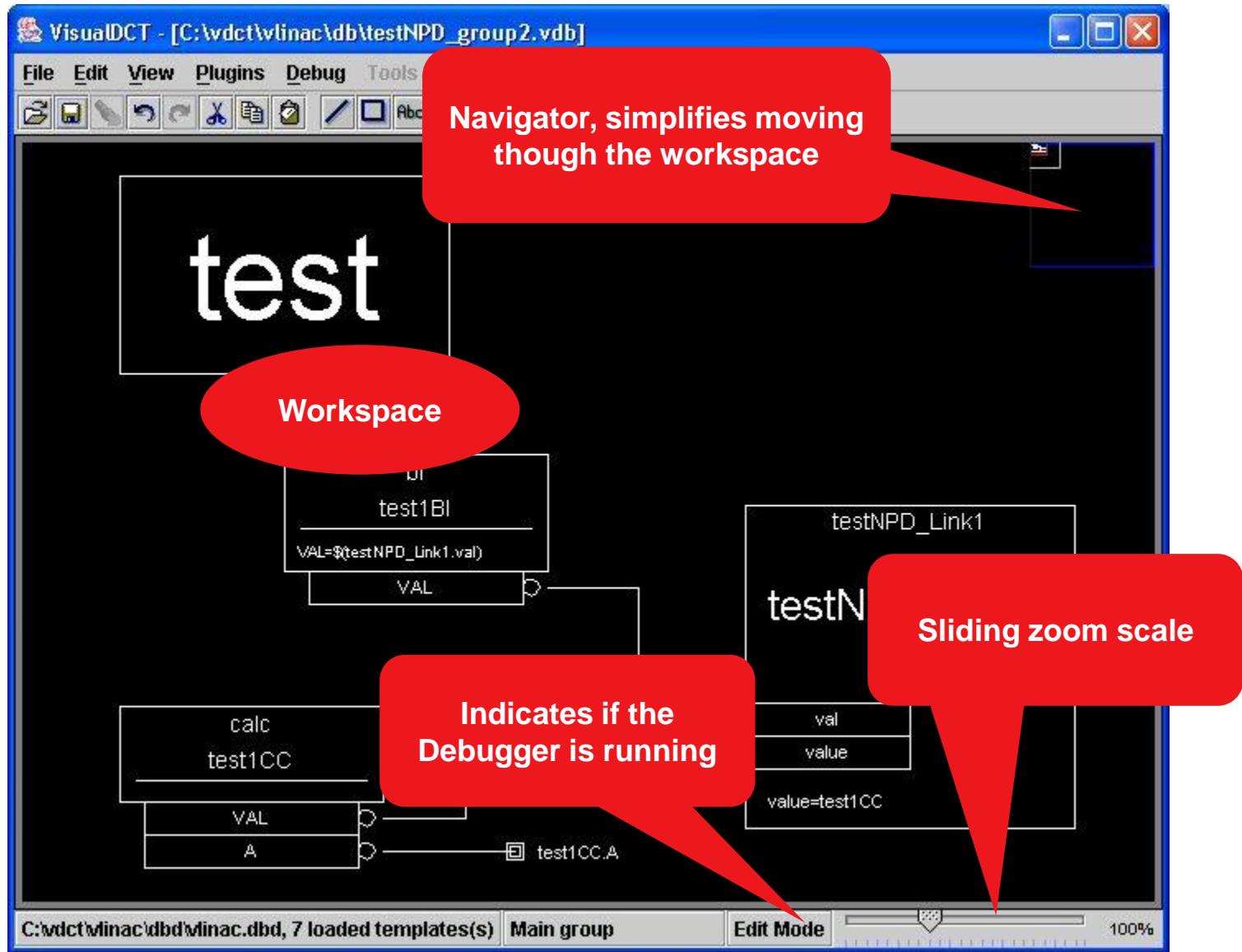
Visual DCT features (cont...)

- Object inspector
- Simple visual linking e.g. automated wiring, default link destinations
- Supports hierarchical design (macros, ports)
- Powerful DB parser
 - Existing DB support
 - Preserves DB comments, record/field order
 - DBs can edited in other tools or manually

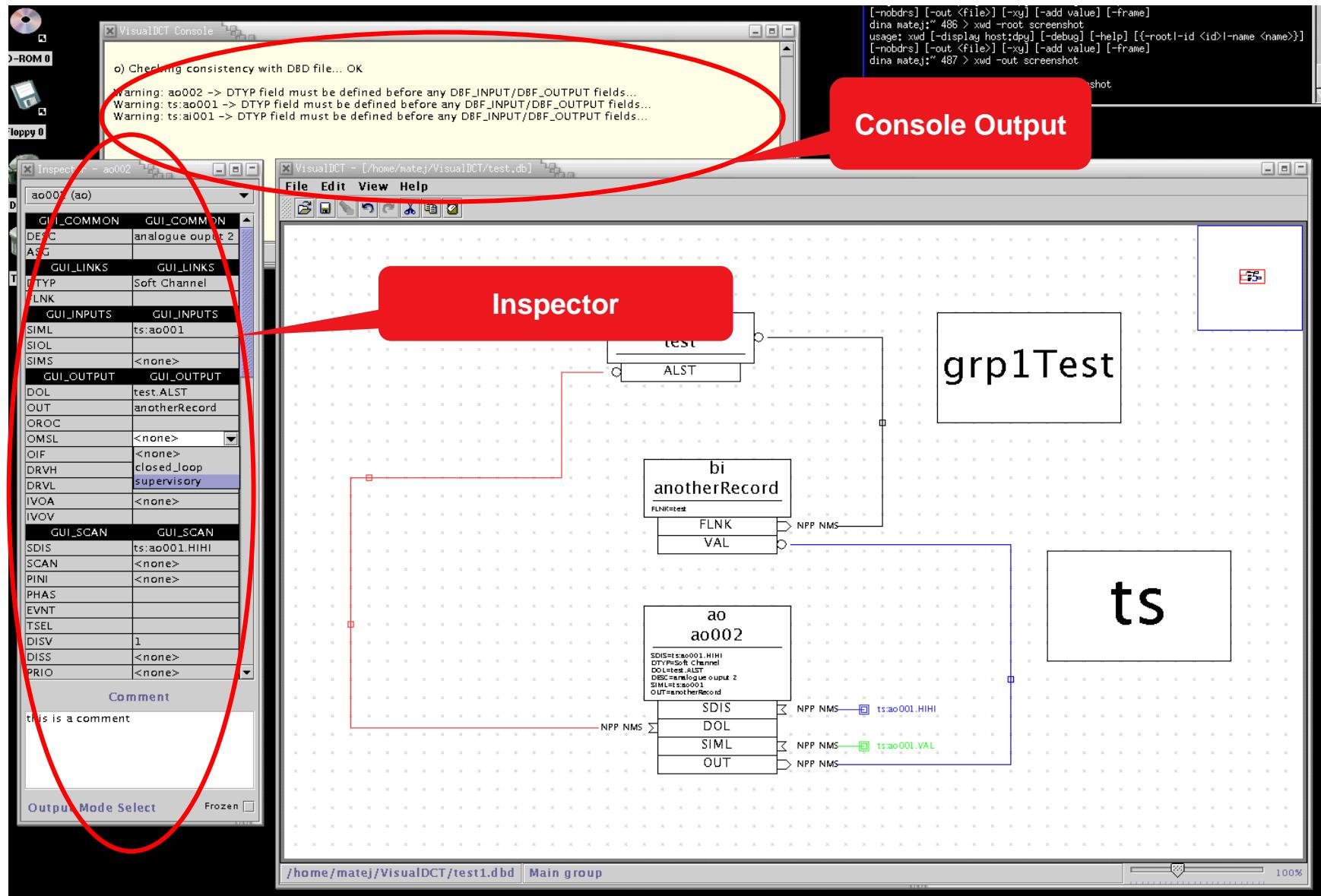
GUI Overview



GUI Overview



GUI Overview



Visual DCT: Create Record

Right Click on
Blank Canvas

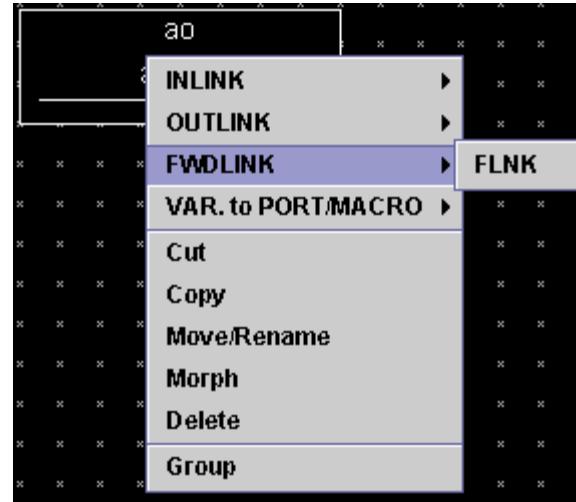
- New record...
- New template instance ▾
- Generate macros...
- New line
- New box
- New textbox
- Template properties...



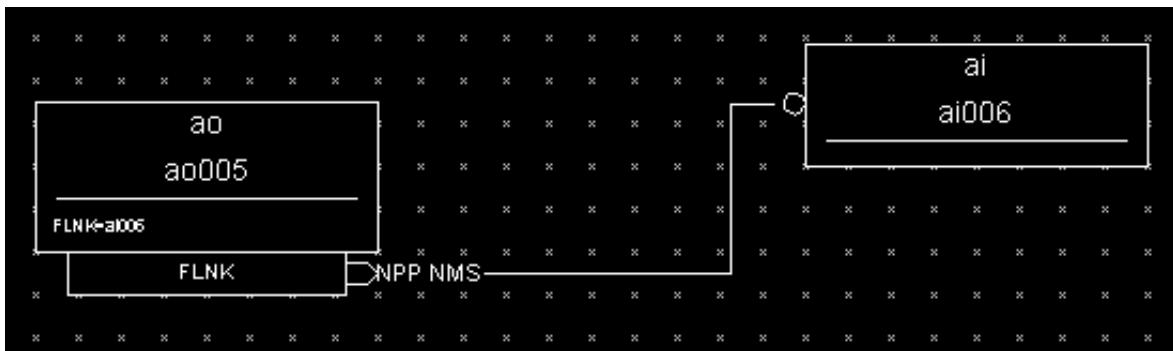
Visual DCT: Linking Records

Process link

Right click on
selected record



Left click on
destination record



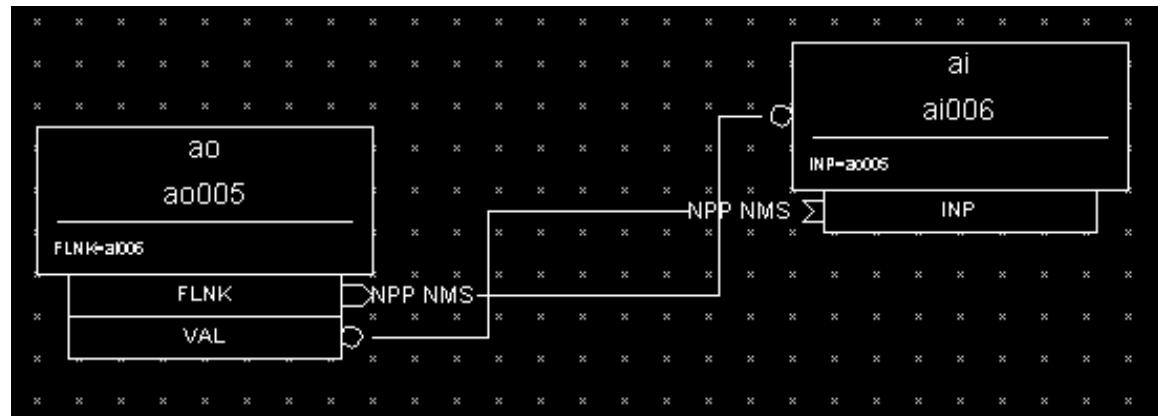
Visual DCT: Linking Records

Input data link 1

Right click on selected record
Select input link
(e.g. .INP)



To select *default* (.VAL) input field
Left click on data source record

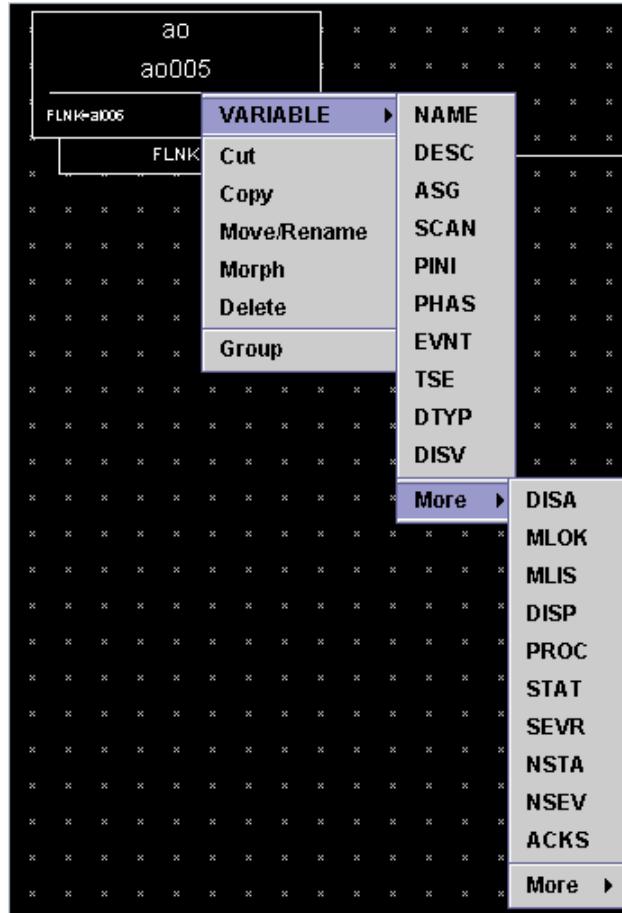


Visual DCT: Linking Records

Input data link 2

To choose a different
data source field
(*not .VAL*)

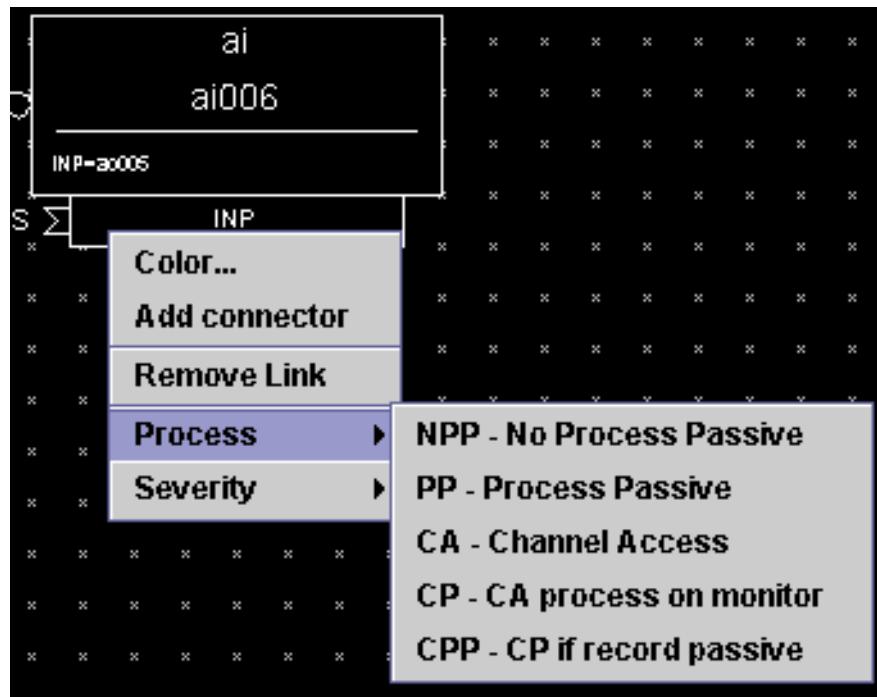
Right click on data
source record and
select field from
menu

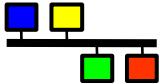


Visual DCT: Linking Records

Link properties

- Right-click on a link to view and set its properties
- *Process* menu allows link process options (PP/CPP etc.) to be set
- *Severity* menu to set Maximise Severity
- *Add connector* creates a 'handle' for manual wire routing

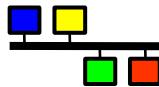




EPICS Database II

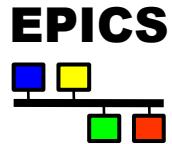
Andy Foster & Philip Taylor

Based on the original by Bob Dalesio, LANL



Outline

- The Record Reference Manual
- Defining EPICS databases
 - Database Definition Files (.dbd)
 - Database files (.db)
- Creating and loading a new database
- How we build a database file with macro substitution
- Input records: ai, bi, mbbi
- Output records: ao, bo, mbbo
- The DOL link and the OMSL field for output records
- The calc record
- The calcout record
- Algorithm and Control Records
 - select, subroutine, fanout, dfanout, sequence
- Database examples



The Record Reference Manual

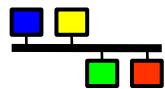
The primary reference document for information about standard EPICS records is the **Record Reference Manual**.

<https://epics.anl.gov/base/R7-0/7-docs/RecordReference.html>

- Chapter 1. Database Concepts (good review)
- Chapter 2. Fields common to *all* records
- Chapter 3. Fields common to *many* records.

A section for each Record Type:

Providing a description of the record's fields and record processing routines for each record type.



Defining EPICS Database Facilities

How does the IOC know what record types and device support options are available ?

- Record types, device support options, enumerated menus, and other configuration options are defined in a “**database definition file**” (**.dbd**)
- During the IOC booting process, one or more **.dbd** files are loaded
- **.dbd** files are created on the workstation to include all the desired definitions for the IOC application. This will normally include standard EPICS record/device support as well as any local or custom support.

Creating Database Definition (.dbd) Files

- Standard **.dbd** files are available from EPICS base but extra definitions must be added for custom device/record support.
- Using *makeBaseApp*, database definition files are defined and built in the **src** directory (*exerciseApp/src*) area.
- Typically, the Makefile in *exerciseApp/src* looks like this:

```
TOP=../..  
include $(TOP)/configure/CONFIG  
DBD += exercise.dbd  
exercise_DBDB += base.dbd  
exercise_DBDB += xxxRecord.dbd  
exercise_DBDB += xxxSupport.dbd  
include $(TOP)/configure/RULES
```

- This creates “*exercise.dbd*” which is placed in the top-level “**dbd**” directory.

A typical database definition (.dbd) file

```

recordtype(ai) {
    field(NAME,DBF_STRING) {
        prompt("Record Name")
        special(SPC_NOMOD)
        size(29)
    }
    field(DESC,DBF_STRING) {
        prompt("Descriptor")
        promptgroup(GUI_COMMON)
        size(29)
    }
    field(ASG,DBF_STRING) {
        prompt("Access Security Group")
        promptgroup(GUI_COMMON)
        special(SPC_AS)
        size(29)
    }
    field(SCAN,DBF_MENU) {
        prompt("Scan Mechanism")
        promptgroup(GUI_SCAN)
        special(SPC_SCAN)
        menu(menuScan)
        interest(1)
    }
    ...
}

recordtype(ao) { ... }
recordtype(bi) { ... }
recordtype(bo) { ... }
recordtype(calc) { ... }

```

```

menu(menuPriority) {
    choice(menuPriorityLOW,"LOW")
    choice(menuPriorityMEDIUM,"MEDIUM")
    choice(menuPriorityHIGH,"HIGH")
}
menu(menuScan) {
    choice(menuScanPassive,"Passive")
    choice(menuScanEvent,"Event")
    choice(menuScanI_O_Intr,"I/O Intr")
    choice(menuScan10_second,"10 second")
    choice(menuScan5_second,"5 second")
    choice(menuScan2_second,"2 second")
    choice(menuScan1_second,"1 second")
    choice(menuScan_5_second,".5 second")
    choice(menuScan_2_second,".2 second")
    choice(menuScan_1_second,".1 second")
}
...
device(ai,CONSTANT,devAiSoftRaw, "Raw Soft Channel")
device(ai,BITBUS_IO,devAilObug, "Bitbus Device")
device(ao,CONSTANT,devAoSoftRaw, "Raw Soft Channel")
device(ao,VME_IO,devAoAt5Vxi, "VXI-AT5-AO")
device(bi,VME_IO,devBiAvme9440, "AVME9440 I")
device(bi,AB_IO,devBiAb, "AB-Binary Input")

...
driver(drvVxi)
driver(drvMxi)
driver(drvGpib)
driver(drvBitBus)

function(myFunction)

```

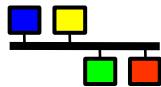
Defining your EPICS Database

*How does an IOC know about the record **instances** (the user's database) ?*

- Record instances are defined in “**database files**” (**.db**) which are built in the **Db** directory of the application area i.e. *exerciseApp/Db*
- During the IOC booting process, one or more **.db** files are loaded
- **.db** files are created on the workstation to include the desired records for the IOC application.

Creating Database (.db) Files

- Since the database file is a simple ASCII file, it can be generated by numerous applications ... as long as the syntax is correct.
 - Text editor
 - Script
 - EPICS-aware Database Configuration Tools
 - Capfast (a schematic electronic wiring application)
 - TDCT (TRIUMF DCT compatible with Capfast)
 - **VDCT**
- An EPICS-aware tool like TDCT/VDCT will read the **.dbd** file and provide menu selections of enumerated fields. It should also detect some database errors/inconsistencies prior to the boot process
- A graphical tool is helpful to document and support hierarchical databases, particularly those with large numbers of links.



A typical database (.db) file

```
record(bo,"$(user):gunOnC") {  
    field(DESC,"Controls e-gun")  
    field(DTYP,"Soft Channel")  
    field(ZNAM,"Beam Off")  
    field(ONAM,"Beam On")  
}  
record(ao,"$(user):cathodeCurrentC") {  
    field(DESC,"set cathode current")  
    field(DTYP,"Raw Soft Channel")  
    field(SCAN,"1 second")  
    field(OROC,".5")  
    field(PREC,"2")  
    field(EGU,"Amps")  
    field(DRVH,"20")  
    field(DRVL,"0")  
    field(HOPR,"20")  
    field(LOPR,"0")  
}
```

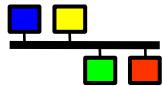
```
record(calc,"$(user):rampM") {  
    field(CALC,"A>6.27?0:A+.1")  
    field(SCAN,"1 second")  
    field(INPA,"$(user):rampM.VAL NPP NMS")  
}  
record(calc,"$(user):cathodeTempM") {  
    field(DESC,"Measured Temp")  
    field(SCAN,"1 second")  
    field(CALC,"C+(A*7)+(SIN(B)*3.5)")  
    field(INPA,"$(user):cathodeCurrentC.OVAL NPP NMS")  
    field(INPB,"$(user):rampM.VAL NPP NMS")  
    field(INPC,"70")  
    field(EGU,"degF")  
    field(PREC,"1")  
    field(HOPR,"200")  
    field(LOPR,"50")  
    field(HIHI,"180")  
    field(LOLO,"130")  
    field(HIGH,"160")  
    field(LOW,"140")  
    field(HHSV,"MAJOR")  
    field(HSV,"MINOR")  
    field(LLSV,"MAJOR")  
    field(LSV,"MINOR")  
}
```

Loading Database Files into the IOC

- Example of a typical startup script (stexercise1.src) file:

```
dbLoadDatabase ("dbd/exercise.dbd")
dbLoadRecords ("db/exercise1.db", "user=ajf67")
iocInit
```

- A database definition file (**.dbd**) is loaded onto the IOC, by calling **dbLoadDatabase** (confusing name)!
 - One **.dbd** file produced at build time
- Databases files (**.db**), can then be loaded, using calls to **dbLoadRecords**.
 - Records must have already been defined in the **.dbd** file.
- Macros within the database files (e.g. \$(user)) can be specified at boot time.
 - Allows same database to be used several times.
 - Not done at Diamond, all macros resolved at build time as follows...



How we build a database file with Macro Substitution

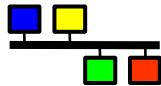
- Use MSI
 - Macro Substitution and Include Tool
 - An EPICS extension
- Invoked as part of the build in the “Db” directory
 - Takes a list of include directories in which to look for template files
`-I/dls_sw/prod/R3.14.12.3/support/tpmac/3-10dls17/db -I...`
 - Takes a “substitutions file” as an argument
-SBL19I-MO-IOC-07.substitutions
- Produces BL19I-MO-IOC-07.db
- Set-up as a “Make” rule in the “Db” directory

MSI Substitutions file

Inside the “Db” directory of the application, we would have “*BL19I-MO-IOC-07.substitutions*” containing:

```
file writeGPIO.template
{
    pattern{ P,                      PORT,      MVAR,   ZNAM,   ONAM }
        { BL19I-MO-STEP-07, BRICK7_S, M36,     OFF,     ON      }
        { BL19I-MO-STEP-07, BRICK7_S, M37,     OFF,     ON      }
        { BL19I-MO-STEP-07, BRICK7_S, I51,     OFF,     ON      }
        { BL19I-MO-STEP-07, BRICK7_S, P1901,  MANUAL, AUTO   }
}
```

Refers to a template file “`writeGPIO.template`” found in one of the include directories.



Binary Input Record

bi	
VentValve	
DTYP=Hy8001	
INP=#C1 S23	
SCAN=I/O Intr	
ZNAM=Closed	
ONAM=Open	
ZSV=NO_ALARM	
OSV=MAJOR	

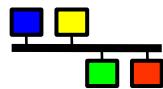
- Name: “VentValve”
- Reads a bit from Hytec 8001 DIO card, Card number 1, Signal 23
- Processed whenever value changes (Interrupt enabled)
- 0 = “Closed”, 1 = “Open” (state strings)
- MAJOR alarm triggered when valve opens

Multi-Bit Binary In (mbbi) Record

mbbi exampleMbbi

DTYP=Raw Soft Channel
ZRVL=0x3
ONVL=0x5
TWVL=0xc
ZRST=Off
ONST=On
TWST=Set @ Default
ZRSV=NO_ALARM
ONSV=NO_ALARM
TWSV=NO_ALARM
UNSV=MINOR

- 0-15 states can be defined:
- State strings: ZRST, ONST, TWST,...
- State values: ZRVL, ONVL, TWVL,...
- State alarm severity: ZRSV, ONSV, TWSV,...
- Undefined value alarm severity is defined in UNSV field
- DTYP = <some hardware support>
- INP: field specifies starting bit number
- NOBT: Number of adjacent bits to be read
- DTYP = Raw Soft Channel
- Read value from input link into RVAL
- Lookup RVAL in list of state values
- Set VAL = Index of state value which matches
 - e.g. if RVAL = 5, VAL = 1 in this example
- When read as string, VAL will be “On” (ONST)
- If no match found in state values, VAL = -1,
 - (string = Illegal Value)
- DTYP = Soft Channel
- Read value from INP directly into VAL

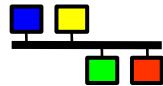


Analogue Input Record

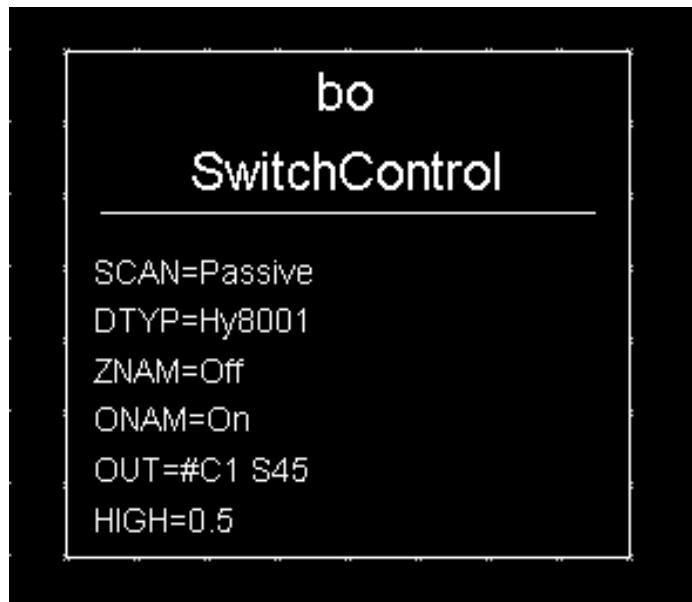
ai Temperature

INP=#C0 S0
DTYP=Hy8401ip
SCAN=.1 second
LINR=LINEAR
EGUF=120
EGUL=0
EGU=Celsius
HIGH=100
LOW=10
HIHI=110
LOLO=5
HHSV=MAJOR
LLSV=MAJOR
HSV=MINOR
LSV=MINOR

- Name: "Temperature"
- Reads from the Hytec 8401 ADC Card 0 Signal 0
- Gets a new value every 0.1 seconds
- Engineering Units (EGU) are Deg Celsius
- LINR field specifies conversion: LINEAR, NO CONVERSION or the name of breakpoint table
- The example shows linear conversion from full ADC range to 0..120 deg Celsius (defined in EGUL, EGUF)
- Four alarm levels and severities :
 - HIHI 110 MAJOR Alarm (HHSV)
 - HIGH 100 MINOR Alarm (HSV)
 - LOW 10 MINOR Alarm (LSV)
 - LOLO 5 MAJOR Alarm (LLSV)
- A weighted average can be built in for smoothing by using the field SMOO, as follows:
- $\text{VAL(new)} = \text{VAL(new)} * (1-\text{SMOO}) + \text{VAL(previous)} * \text{SMOO}$



Binary Output Record



- Name: “SwitchControl”
- Writes a bit to Hytec 8001 Digital I/O card, Card number 1, Signal 45
- Processed whenever new value entered from GUI
- 0 = “Off”, 1 = “On” (state strings)
- Optionally, the output signal can be held high (1) for a specified number of seconds, defined by the field HIGH.

Multi-Bit Binary Output (mbbo) Record

mbbo exampleMbbo

ZRVL=0x3
ONVL=0x5
TWVL=0xc
ZRST=Off
ONST=On
TWST=Set @ Default
UNSV=MINOR
DTYP=Raw Soft Channel
ZRSV=NO_ALARM
ONSV=NO_ALARM
TWSV=NO_ALARM

- 0-15 states can be defined:
- State strings: ZRST, ONST, TWST,...
- State values: ZRVL, ONVL, TWVL,...
- State alarm severity: ZRSV, ONSV, TWSV,...
- Undefined value alarm severity is defined in the UNSV field

DTYP = <some hardware support>

OUT: field specifies starting bit number for write
NOBT: Number of adjacent bits to write

DTYP = Raw Soft Channel

VAL is used as an index into the states

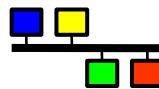
RVAL is set equal to value assigned to that state

e.g. if VAL = 2, then RVAL = 0xc = 12 in this example

If no match found: RVAL = 0.

DTYP = Soft Channel

Write VAL directly to the OUT link



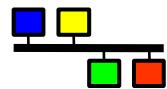
ao	
TempControl	
DTYP=Hy8402ip	
SCAN=1 second	
OUT=#C1 S3	
OROC=1.0	
OIF=Incremental	
DRVH=110	
DRVL=5	
IVOA=Set output to IVOV	
IVOV=20	
HOPR=110	
LOPR=5	
LINR=LINEAR	
EGUF=120	
EGUL=0	
HIHI=110	
LOLO=5	
HIGH=100	
LOW=10	
HHSV=MAJOR	
LLSV=MAJOR	
HSV=MINOR	
LSV=MINOR	
HYST=0.1	
PREC=2	
EGU=Celsius	

Analogue Output Record

- Many fields similar to AI record
- Extra fields: OIF = Incremental or Full output
OROC = Output Rate of Change
- After a new value is computed, it is limited by DRVH and DRVL.
These are software drive limits on the output
- The HOPR and LOPR fields set the upper and lower display limits. They must be in the range
 $DRV\text{L} \leq LOPR \leq HOPR \leq DRV\text{H}$
- IVOA specifies action to take when record is put into INVALID alarm severity
 - IVOA = 'Set output to IVOV',
• IVOV = 20 in this case
- The Alarm deadband (HYST) specifies that the alarm state should only change when the value changes by more than 0.1 deg.
- The values will be displayed (PREC) with 2 decimal places.

The DOL and OMSL fields

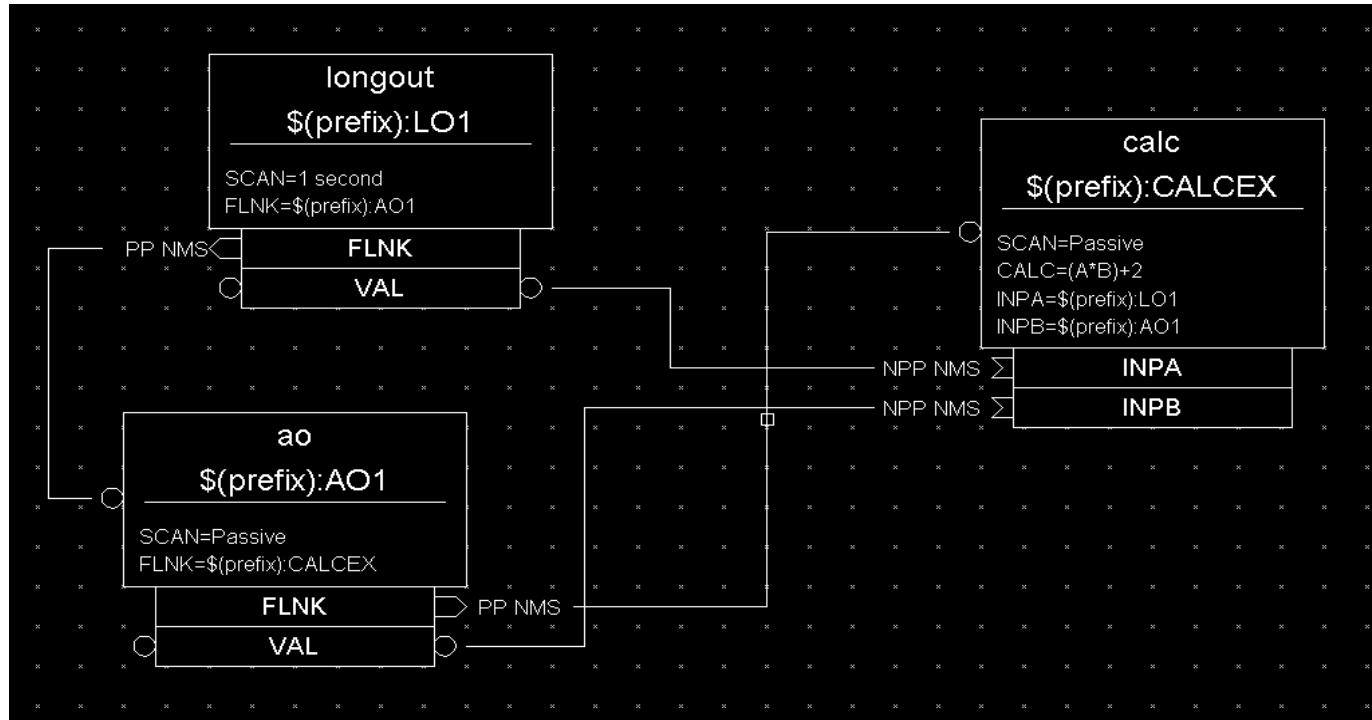
- All EPICS **output** records have these two fields:
 - DOL (Desired Output Location)
 - OMSL (Output Mode Select)
- DOL is a confusing name, it's actually an **input link!**
- When OMSL = *closed_loop*
 - DOL is used to fetch the value from where it is linked
- When OMSL = *supervisory*
 - DOL is NOT used. In this case we can write the value to VAL over Channel Access
 - Whatever value is in VAL is output on the OUT link.
 - **WARNING:** The default value for OMSL is *supervisory*. A common error is to forget to set it to *closed_loop* when you want to use the DOL link.



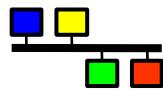
The calc record

- 12 inputs link fields INPA through INPL
- Values stored in fields A - L (can write to these if input links not used)
- Algebraic Operators: ABS, SQR, MIN, MAX, CEIL, FLOOR, LOG, LOGE, EXP, ^, **, +, -, *, /, %, NOT
- Trigonometric Operators: SIN, SINH, ASIN, COS, COSH, ACOS, TAN, TANH, ATAN
- Relational Operators: >=, >, <=, <, #, = [*Last 2 not "C" syntax "!=" "=="*]
- Logical Operators: &&, ||, ! (Not)
- Bit-wise Operators: |, &, OR, AND, XOR, ~, <<, >>
- Parentheses and nested parentheses are supported
- The C '?' operator is supported for conditionals (expr)?(true):(false)

Database - Calculation



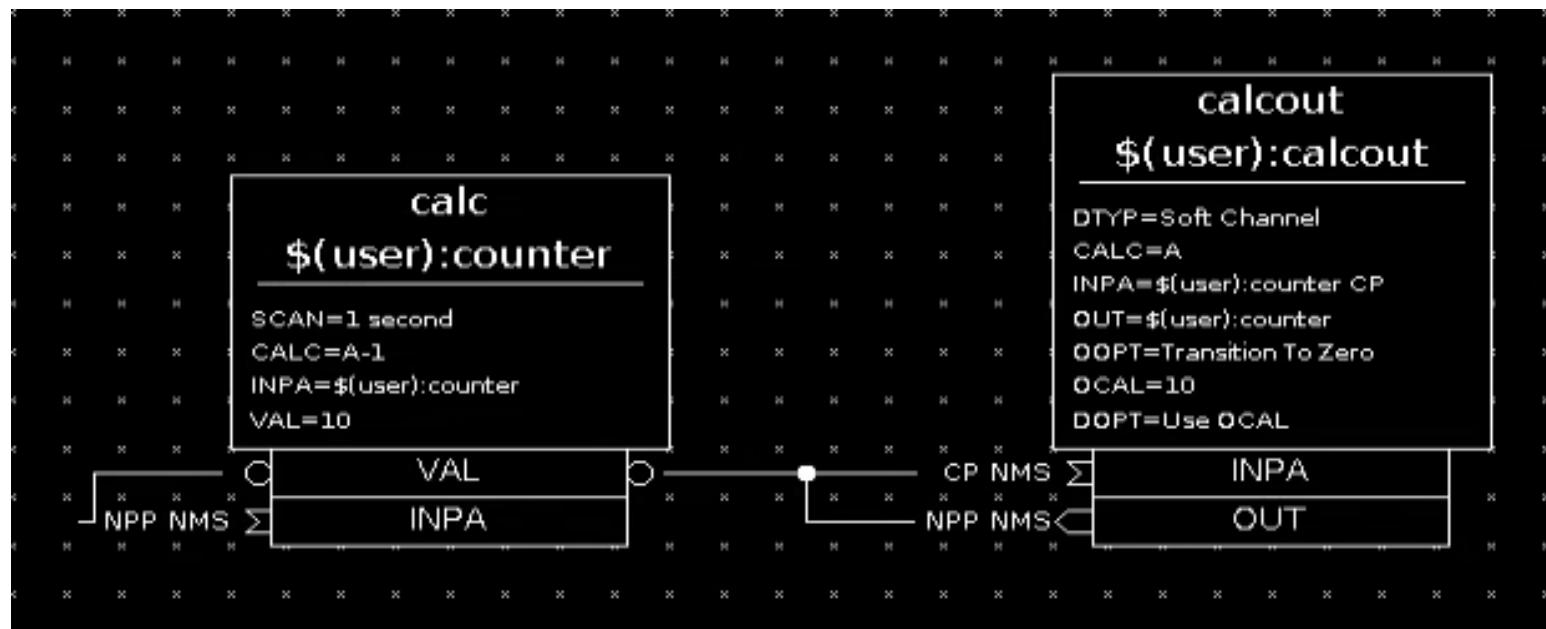
- In this example, the Calculation record gets the value of field A (via INPA) from a longout record (LO1) and field B from an analogue output record (AO1)
- The calculation expression $(A * B) + 2$ is defined in the calculation record's CALC field
- The result (in the VAL field of the Calculation record) will be the value in the VAL field of record LO1 multiplied by the value of the VAL field of record AO1, plus 2.0



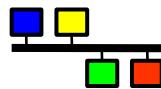
The calcout record

- Like a calc record in all ways except it has 5 extra, useful, fields:
- OCAL: Another field to store a calculation
- OUT: Link field to push a value to another record
- OOPT: This field controls **when** the OUT link fires
 - "Every Time"
 - "On Change"
 - "When Zero"
 - "When Non-zero"
 - "Transition To Zero"
 - "Transition To Non-zero"(The result of the CALC expression is used)
- DOPT: Which data is put on the OUT link? ("Data Output Field")
 - If(DOPT = "Use CALC") OUT pushes the result of the CALC calculation
 - If(DOPT = "Use OCAL") OUT pushes the result of the OCAL calculation
- ODLY = number of seconds to wait before the output is sent
(The record has PACT set TRUE during this period)

calcout - resetting a counter

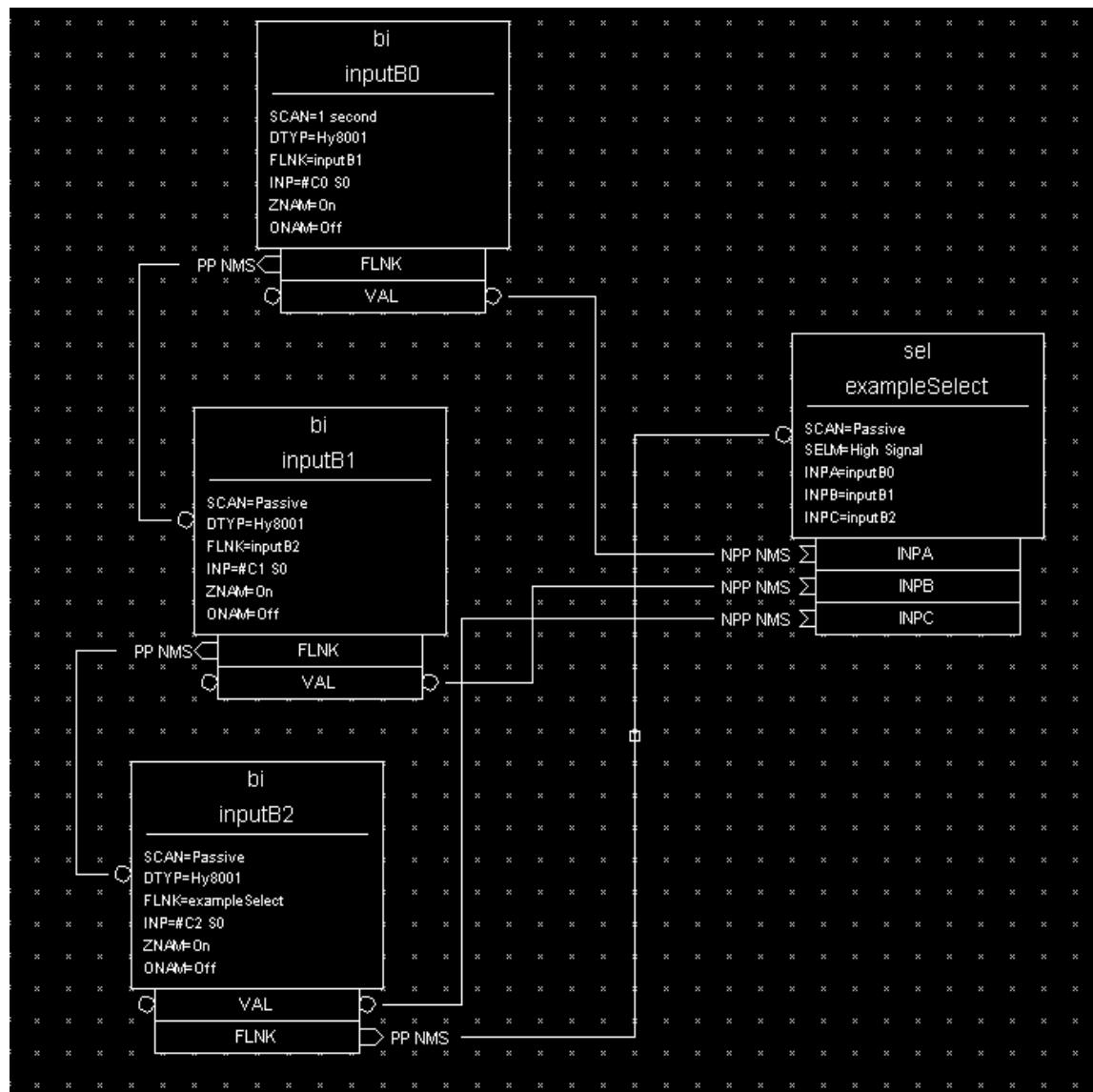


- The CALC record is counting down from 10...
- The input CP link on the CALCOUT record means it will process whenever the value of the CALC changes i.e. it will also process every second
- When the CALC VAL field reaches 0, the CALCOUT will notice the “Transition To Zero” through the result of the CALC expression (“A”) and fire its OUT link.
- The value on the OUT link is the result of the calculation in OCAL (the constant 10).
- This is written to the VAL field of the CALC record which resets the counter.



Algorithm/Control: Select Record

12 input links, four select options



- ❖ In the example shown, 3 BI records read in a bit (Signal 0) from 3 different Hytec 8001 cards (C0, C1 and C2)
- ❖ The Select record has its **SELM** field set to “*High Signal*” = select the highest value of the fields read in on its input links
- ❖ **SELM** can be:
 - ❖ *Specified (use SELN field)*
 - ❖ *High Signal*
 - ❖ *Low Signal*
 - ❖ *Median Signal*
- ❖ In this example, the Select record VAL field will take the value 1 whenever *any* of the input bits are 1.

Algorithm/Control: Subroutine Record

Attach user provided subroutines

```
* * * * * * * * * * *  
sub  
exampleSub  
_____  
SCAN=.1 second  
INAM=subInit  
SNAM=subProcess  
BRSV=MAJOR  
* * * * * * * * * *
```

- The subroutine record has 12 input links (**INPA** - **INPL**) but only one output value (**VAL**). All are of type **DOUBLE**
- Every subroutine record must have an associated *initialisation* routine (defined in the **INAM** field) and a *process* routine (defined in the **SNAM** field)
- An error return value (< 0) from the subroutine can set an alarm (use **BRSV** field)

Example Subroutine Record code

```

#include <stdio.h>
#include <dbDefs.h>
#include <subRecord.h>
#include <registryFunction.h>

static long subInit(psub)
    struct subRecord *psub;
{
    printf("subInit was called\n");
    return(0);
}

static long subProcess(psub)
    struct subRecord *psub;
{
    psub->val++;
    return(0);
}

epicsRegisterFunction(subInit);
epicsRegisterFunction(subProcess);

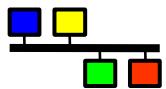
```

- In the example shown, the initialisation routine (defined in the **INAM** field as **subInit**) just outputs a text message
- The process routine (defined in the **SNAM** field as **subProcess**) accesses record fields via a pointer to the *subRecord* structure. In the example, it simply increments the **VAL** field each time it is called
- The functions must be registered using EPICS registry functions
- The registered functions must also be declared in **function()** statements in the **.dbd** file:

```

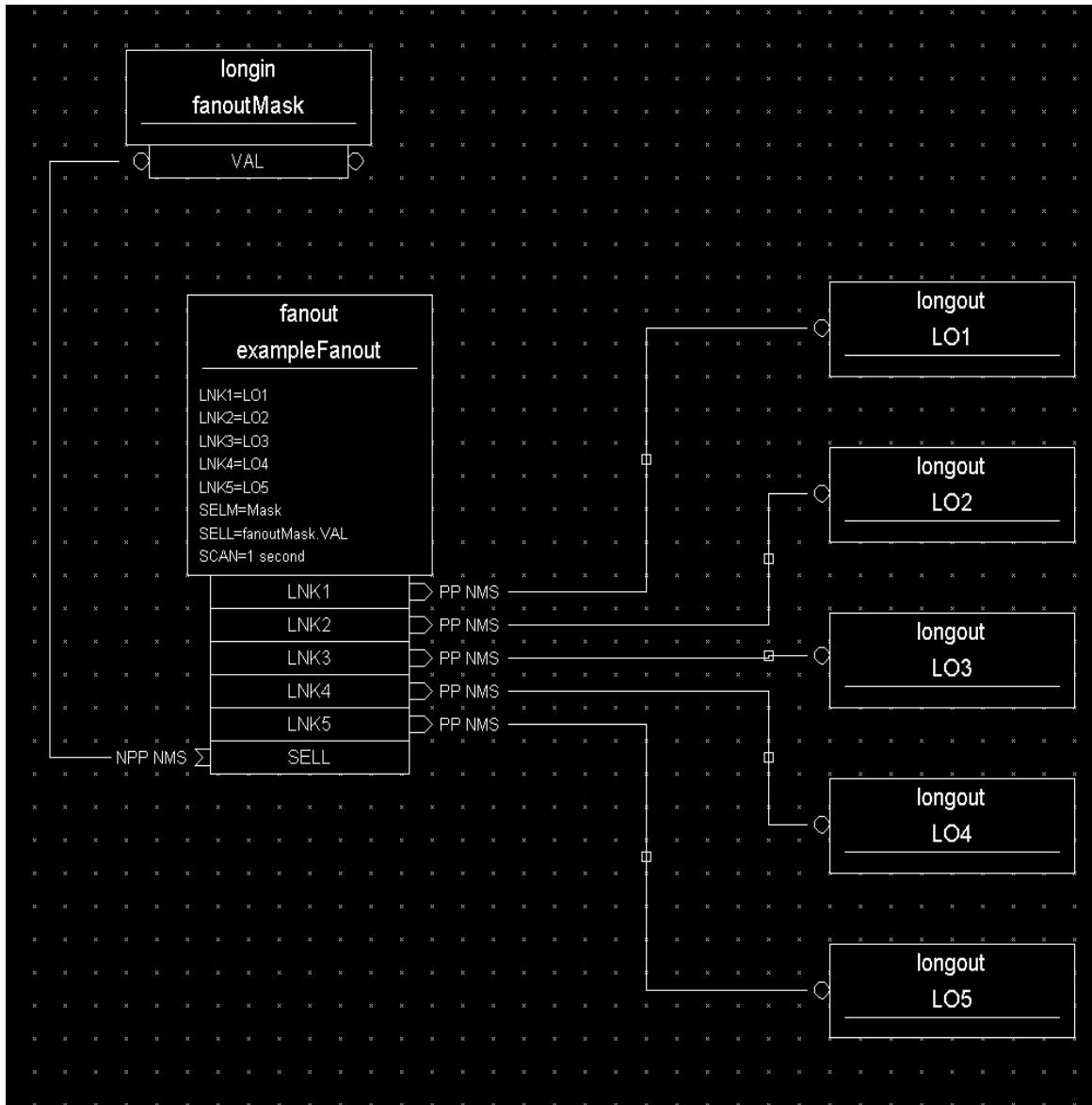
function (subInit)
function (subProcess)

```

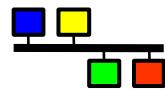


Algorithm/Control: Fanout Record

Forward links with no data exchange

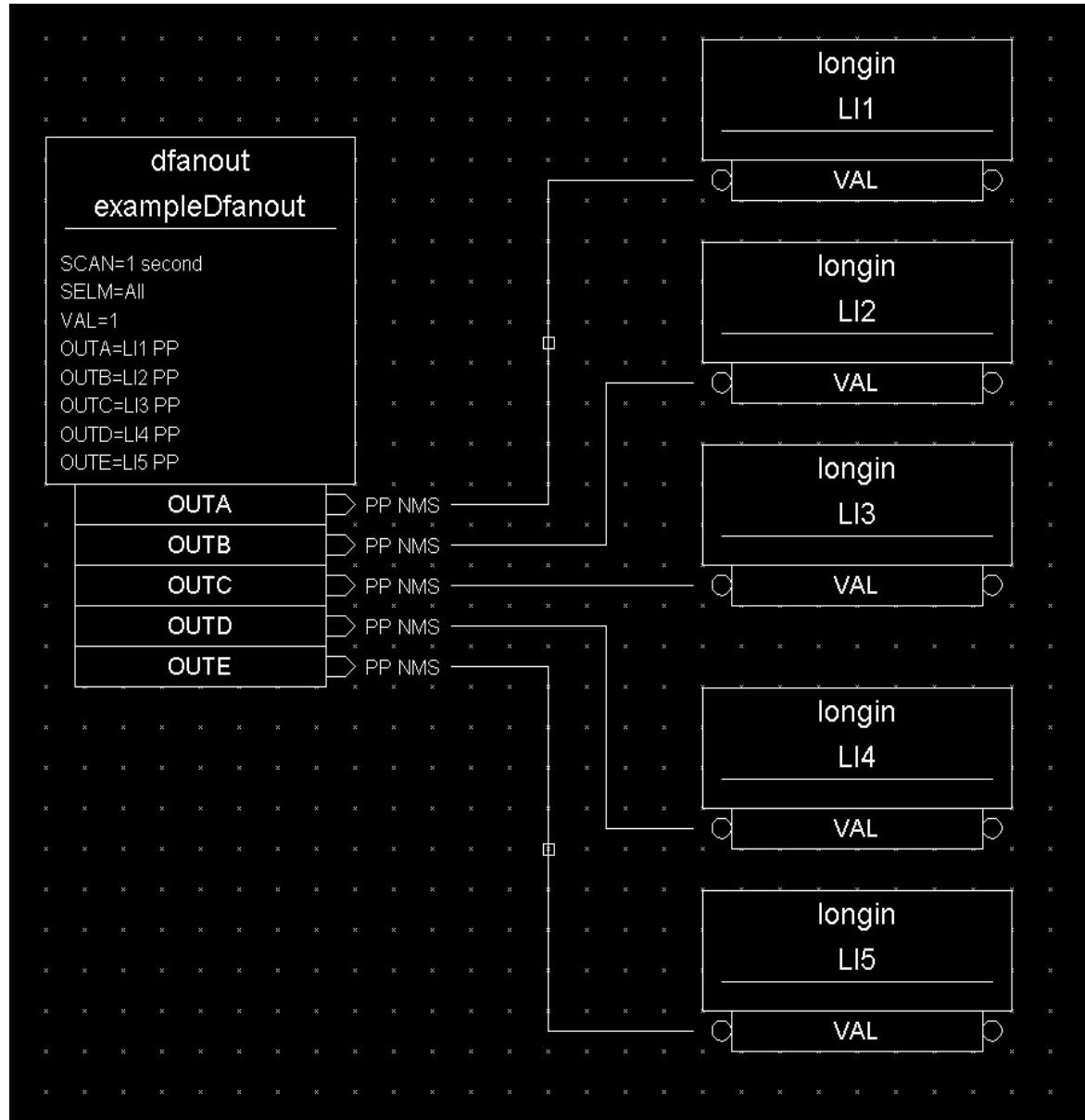


- Used to fanout processing, this record can trigger up to 6 forward links
- SELM** sets the algorithm for executing the forward links :
 - All**: processes all links in order
 - Specified**: process only the link number set in **SELN**
 - Mask**: process the links by interpreting the value in **SELN** as a bit mask e.g. 5 would be LINK1, LINK3
 $\text{LINK}(n+1), \text{LINK}(m+1) = 2^n + 2^m$
- The link field **SELL** can be used to pull a value into **SELN**.

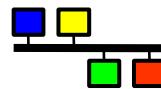


Algorithm/Control: Dfanout Record

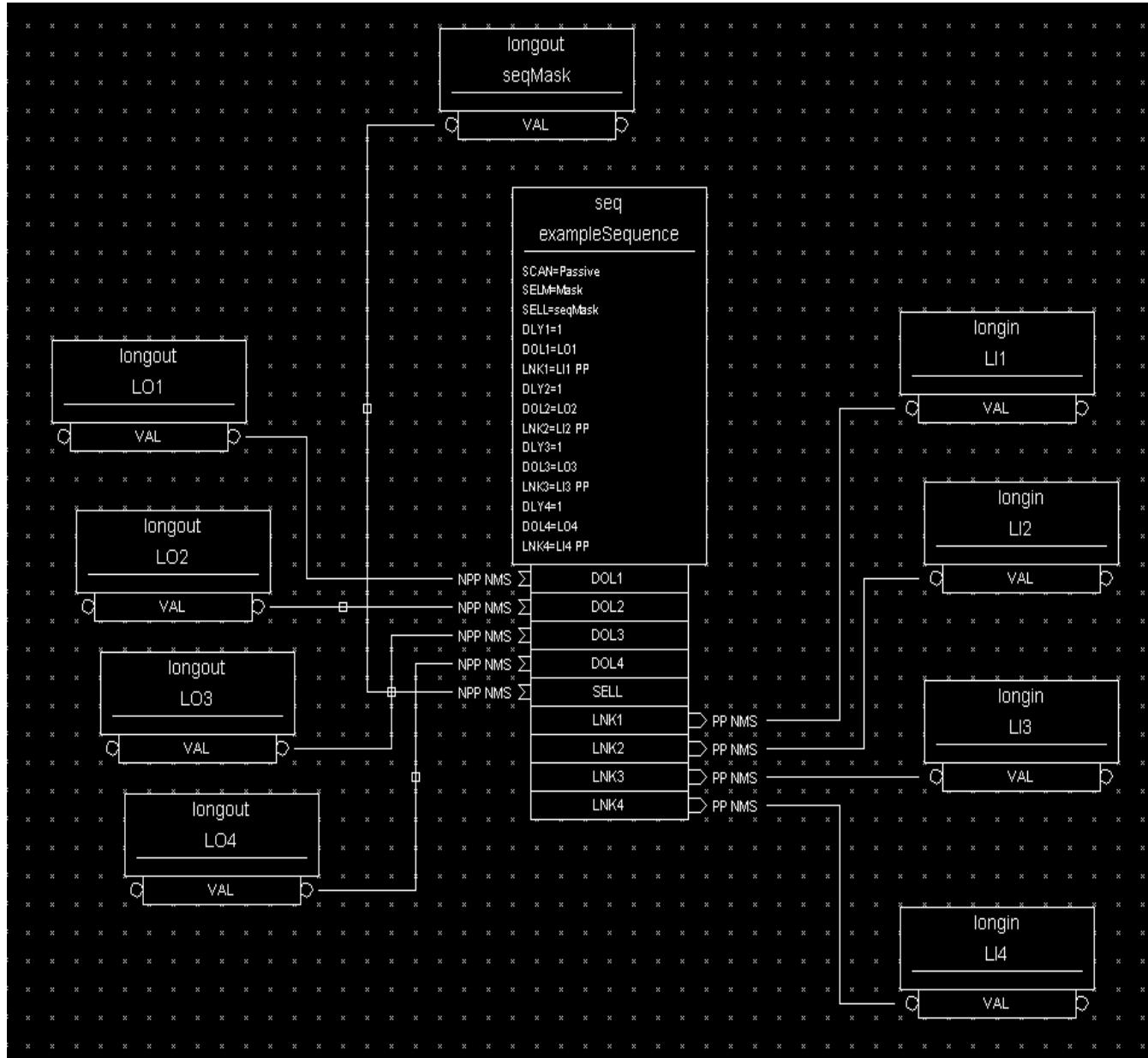
Writes a single source of data to up to eight output links



- Used to fanout the same value to up to 8 database locations.
- Uses **SELM**, **SELN** & **SELL** fields to select data links in the same way as the Fanout record



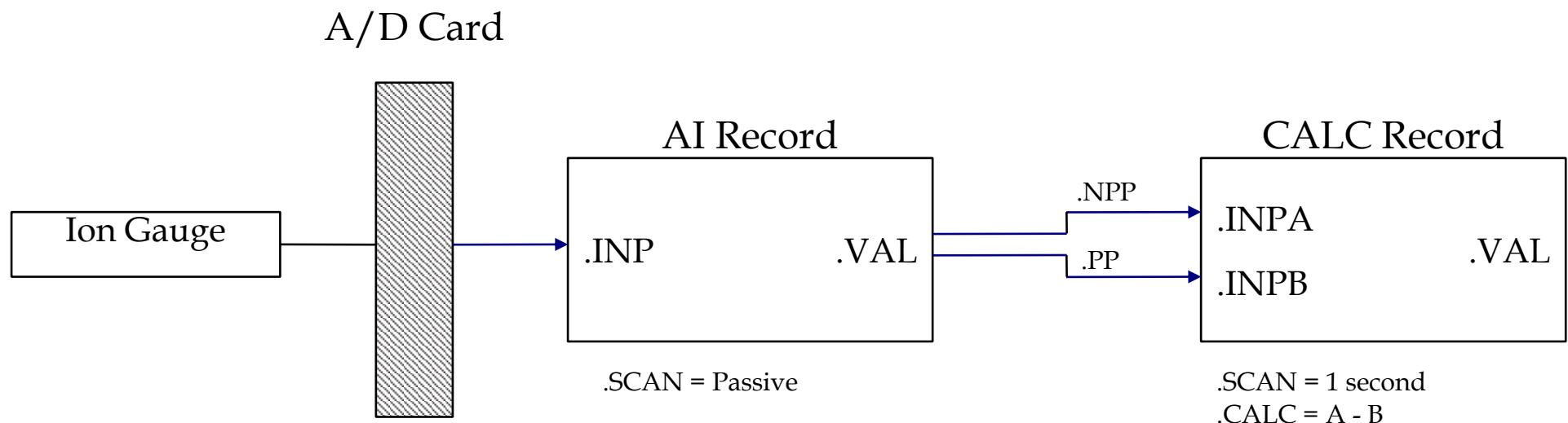
Algorithm/Control: Sequence Record Input/Output link pair execution



- ❖ Process up to ten “Input/Output link” pairs.
- ❖ Specify a delay (**DLYx**) between each link pair execution. The delays for the link pairs are additive.
- ❖ Uses **SELM**, **SELN** & **SELL** fields to select data links in the same way as the Fanout record

Database Examples

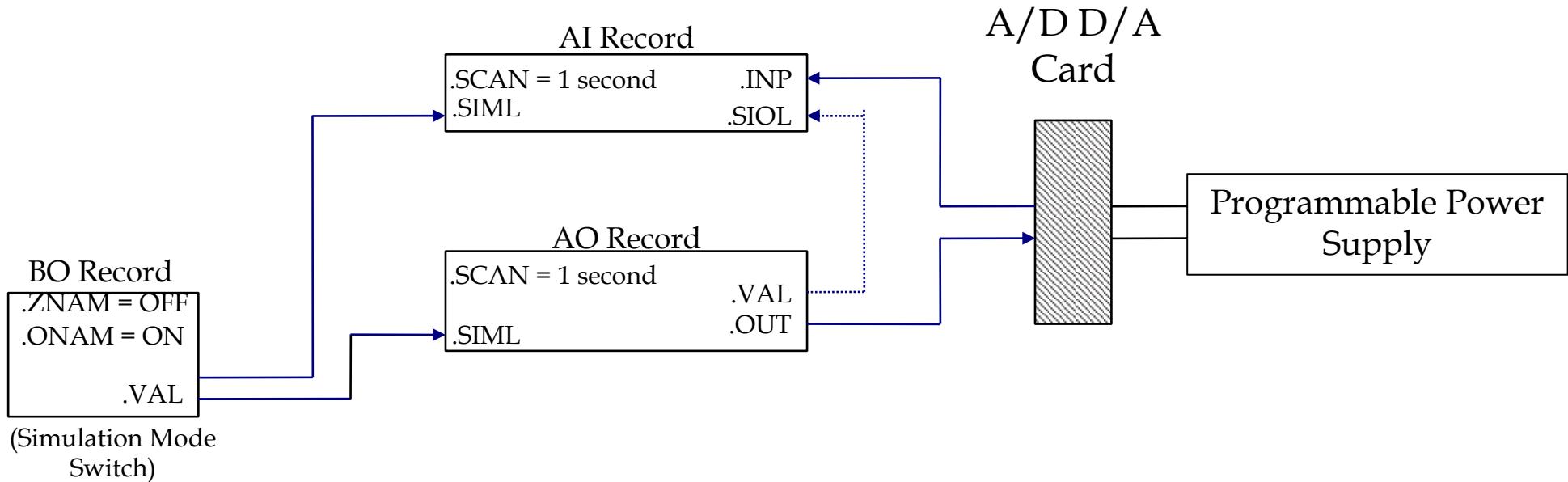
Calculating “Rate-of-Change” of an Input



The CALC record processes its input links in alphabetical order. INPA fetches data that is 1 second old because it does not request processing of the AI record (.NPP). INPB fetches current data because it requests the AI record to process (.PP). The subtraction of these two values reflects the ‘rate of change’ (difference/sec) of the pressure reading.

Database Examples

Simulation Mode



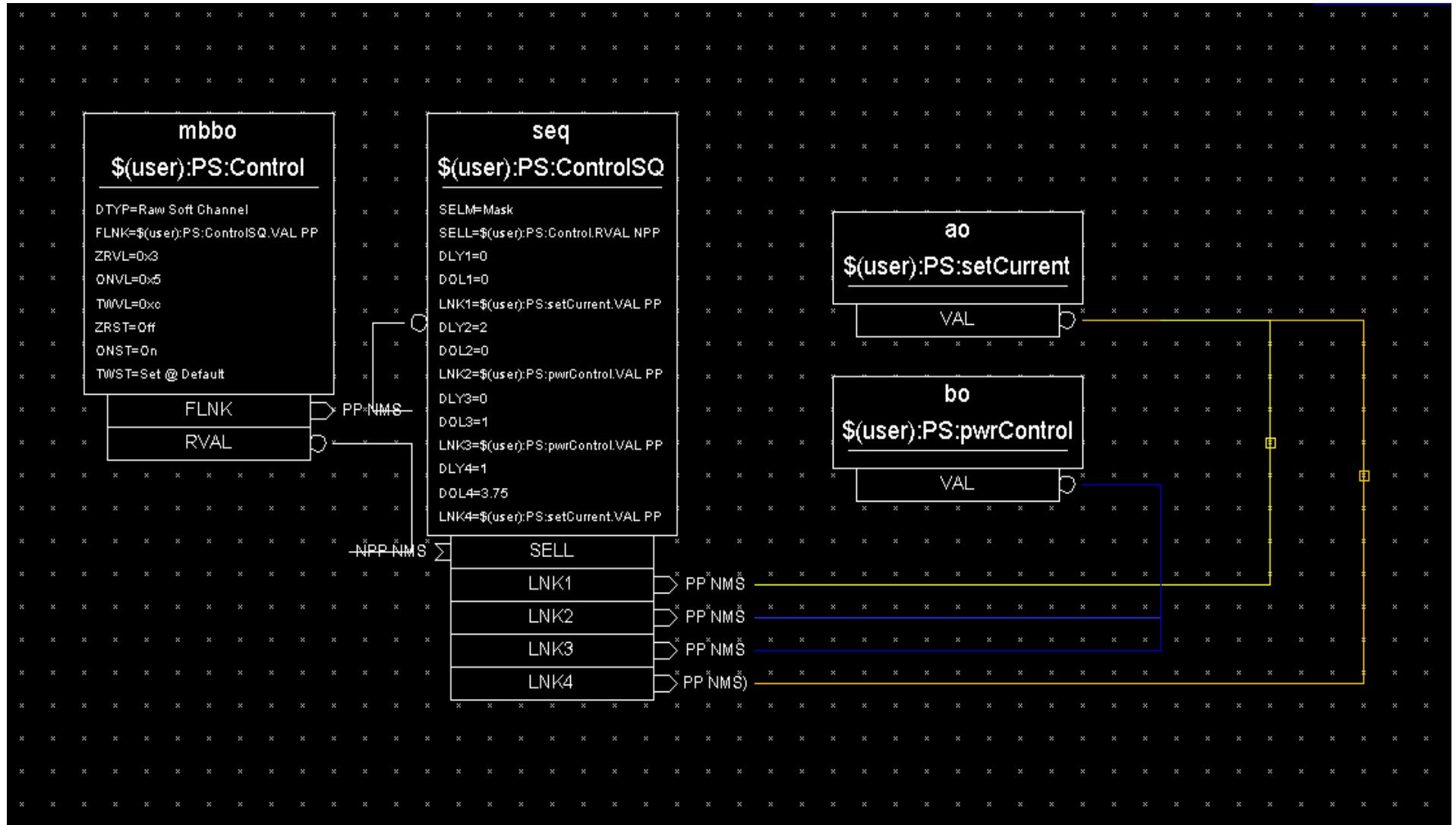
The AI and AO records get their simulation mode (ON/OFF) from the BO record via their .SIML input links.

When in simulation mode, the AO/AI records do not call device support. The AI record fetches its input from the AO record, via the .SIOL (Simulation Input) link field.

When NOT in simulation mode, the AI is reading directly from the power supply. The AO is setting the power supply.

Database Examples

Different Actions Based on Operator Selection - 1



Different links in the sequence record are executed for each selection of the mbbo. This allows much functionality to be specified in only two records.

Database Examples

Different Actions Based on Operator Selection - 2

The **seq** record has its SELM field set to Mask. This means: use the value fetched in via the SELL link as a binary digit to select which output links to fire i.e.

$$\text{SELL} = 2^n + 2^m \Rightarrow \text{fire links } (n+1) \text{ and } (m+1)$$

SELL values from the **mbbo** record, as shown, correspond to:

'Off' = ZRST $\Rightarrow 0x3 \Rightarrow$ Select output links 1 and 2
(*setCurrent=0 immediately + pwrControl=0 after 2 secs.*)

'On' = ONST $\Rightarrow 0x5 \Rightarrow$ Select output links 1 and 3
(*setCurrent=0 immediately + pwrControl=1 after 2 secs.*)

'Set @ Default' = TWST $\Rightarrow 0xc \Rightarrow$ Select output links 3 and 4
(*pwrControl=0 after 2 secs. + setCurrent=3.75 after a further 1 sec.*)

Database Examples

Different Actions Based on Operator Selection - 3

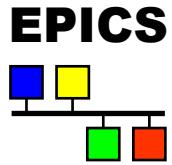
The database definition (.db) file

```

record(mbbo,"$(user):PS:Control") {
    field(DTYP,"Raw Soft Channel") ← Note Device Type
    field(FLNK,"$(user):PS:ControlSQ.VAL PP NMS")
    field(ZRVL,"0x3")
    field(ONVL,"0x5")
    field(TWVL,"0xc")
    field(ZRST,"Off")
    field(ONST,"On")
    field(TWST,"Set @ Default")
}
record(seq,"$(user):PS:ControlSQ") {
    field(SELM,"Mask")
    field(SELL,"$(user):PS:Control.RVAL NPP NMS")
    field(DLY1,"0")
    field(DOL1,"0")
    field(LNK1,"$(user):PS:setCurrent.VAL PP NMS")
    field(DLY2,"2")
    field(DOL2,"0")
    field(LNK2,"$(user):PS:pwrControl.VAL PP NMS")
    field(DLY3,"0")
    field(DOL3,"1")
    field(LNK3,"$(user):PS:pwrControl.VAL PP NMS")
    field(DLY4,"1")
    field(DOL4,"3.75")
    field(LNK4,"$(user):PS:setCurrent.VAL PP NMS")
}

```

When used with a suitable GUI object, the mbbo state string set will be presented automatically in a menu to the user



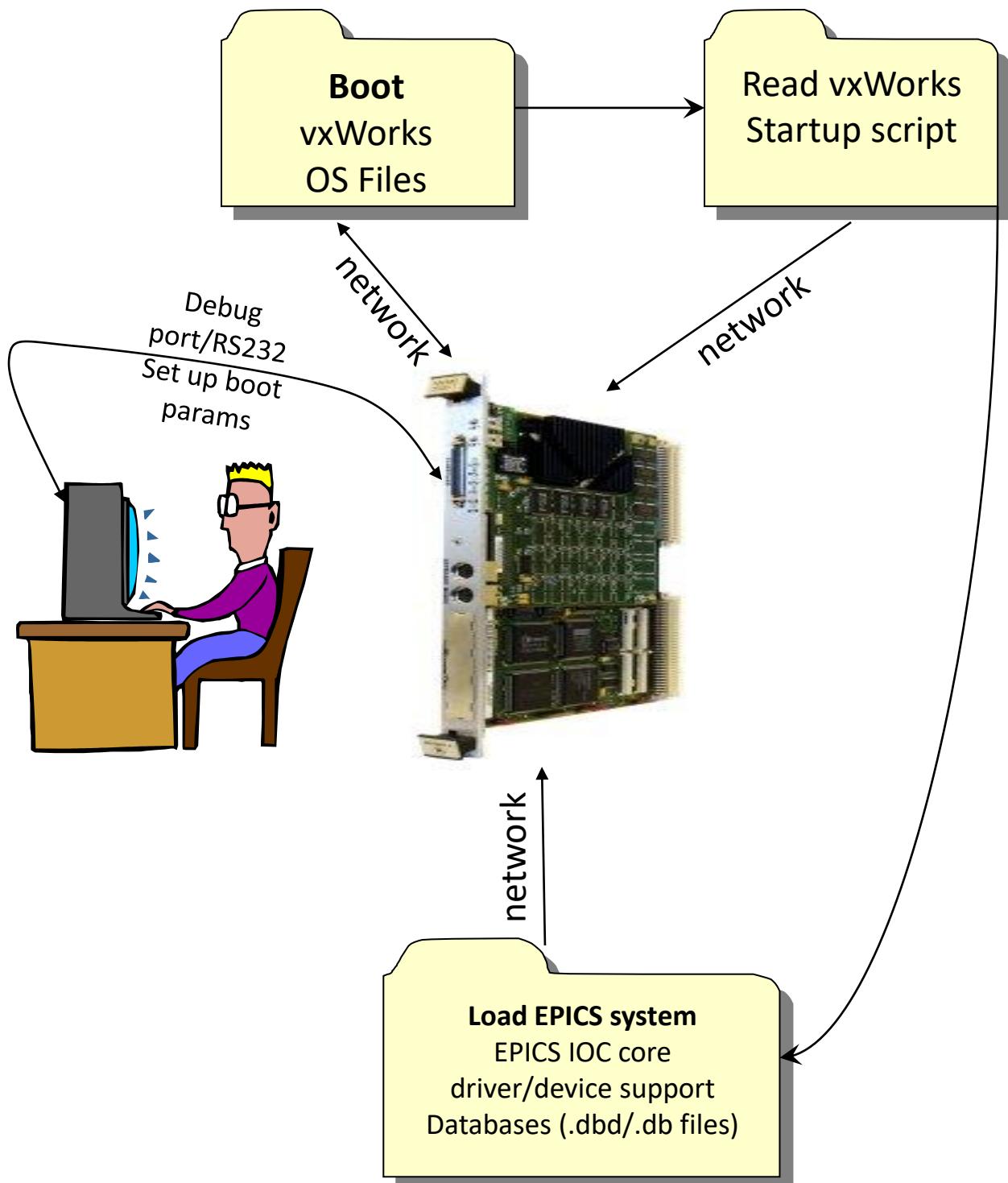
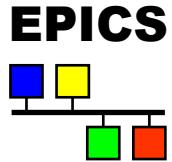
EPICS IOC

Configuration

and Startup

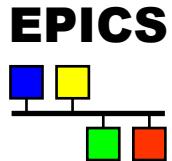
Philip Taylor

Host/Target EPICS Startup (vxWorks)



Linux IOC

Local EPICS Startup



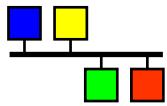
Linux PC

Run built application (binary) from command line with startup script as:

```
../../../../bin/linux-x86_64/stexercise1.boot
```

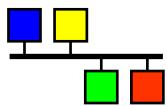
IOC shell runs
EPICS startup script

Load EPICS system
Load EPICS IOC core
device/driver support
databases



Startup Script

- The EPICS IOC shell is a simple command interpreter which provides a limited set of commands. It is used to interpret startup scripts and to execute commands entered at the console terminal.
- You need to include commands for loading and starting of your application-specific executable code, any custom EPICS records or device support, sequence programs and of course the database itself.
- Typical startup script (`stexercise1.src`) is shown below.



Startup script (stexercise1.src)

```
!$(INSTALL)/bin/$(ARCH)/exercise

## You may have to change exercise to something
## else everywhere it appears in this file

cd "$(INSTALL)"

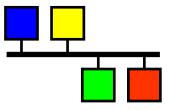
# Load binaries on architectures that need to do so.
# VXWORKS_ONLY, LINUX_ONLY and RTEMS_ONLY
# are macros that resolve to a comment symbol on
# architectures that are not the current build
# architecture, so they can be used liberally to do
# architecture specific things. Alternatively, you can
# include an architecture specific file.
$(VXWORKS_ONLY)Id < bin/$(ARCH)/exercise.munch

## Register all support components
dbLoadDatabase("dbd/exercise.dbd")
exercise_registerRecordDeviceDriver(pdःbase)

## Load record instances
dbLoadRecords("db/sim.db","sim=ajf67")
dbLoadRecords("db/exercise1.db","user=ajf67,sim=ajf67")

iocInit()

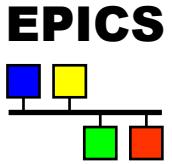
## Start any sequence programs
#seq &controlFilter,"user=ajf67"
```



Some Linux IOC Shell commands

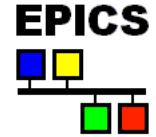
Command	Description
help [command ...]	Display synopsis of specified commands. Wild-card matching is applied so ‘help db*’ displays a synopsis of all commands beginning with the letters ‘db’. With no arguments this displays a list of all commands.
#	A ‘#’ as the first non-whitespace character on a line marks the beginning of a comment, which continues to the end of the line (however some versions of Base may require a space after the ‘#’ character to properly recognize it as a comment)
exit	Stop reading commands. When the top-level command interpreter encounters an exit command or end-of-file (EOF) it returns to its caller.
cd directory	Change working directory to directory.
pwd	Print the name of the working directory.
var [name [value]]	If both arguments are present, assign the value to the named variable. If only the name argument is present, print the current value of that variable. If neither argument is present, print the value of all variables registered with the shell. Variables are registered in application database definitions using the variable keyword as described in Section 6.9 on page 104.
show [-level] [task ...]	Show information about specified tasks. If no task arguments are present, show information on all tasks. The level argument controls the amount of information printed. The default level is 0. The task arguments can be task names or task i.d. numbers.
system command_string	Send command_string to the system command interpreter for execution. This command is present only if some application database definition file contains registrar(iocshSystemCommand) and if the system provides a suitable command interpreter (vx-Works does not).
epicsEnvSet name value	Set environment variable name to the specified value.
epicsEnvShow [name]	If no name is specified the names and values of all environment variables will be shown. If a name is specified the value of that environment variable will be shown.
epicsParamShow	Show names and values of all EPICS configuration parameters.
iocLogInit	Initialize IOC logging.
epicsThreadSleep sec	Pause execution of IOC shell for sec seconds.

Some IOC Test Commands



Note – on Linux iocsh, the “ ” are optional

<i>dbl</i>	<i>List all records</i>
<i>dbgrep "string"</i>	<i>List records with specified string in their name</i>
<i>dbpr "record_name", "interest_level"</i>	<i>Print fields of record (with optional interest level 0-4)</i>
<i>dbgf "pv_name"</i>	<i>Get channel (field) value</i>
<i>dbpf "pv_name", "value"</i>	<i>Put channel (field) value</i>
<i>dbtr</i>	<i>Test record processing</i>
<i>dbnr</i>	<i>Print number of records of each type and total number</i>
<i>dbior "driver_name", "interest_level"</i>	<i>I/O test report</i>
<i>dbhcr</i>	<i>Hardware configuration report: list hardware links</i>



Debugging EPICS Applications

Andy Foster
Observatory Sciences Ltd.

Basic IOC Startup Errors

After every change to an IOC database or code, you should **always** watch the boot-up on the IOC console:

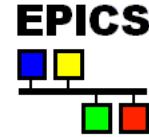
- 1) Were there any command errors when the startup script ran?
- 2) Have the EPICS records been loaded as expected?
 1. Use the **dbl** command on the IOC console to list the records.
- 3) Have all the .dbd and .db files been loaded correctly? File names and locations correct?
- 4) Have the EPICS records got the right names – may have wrong or missing macro value when loading database.
- 5) Device configuration commands ran OK?
- 6) Any errors during IOC initialization (iocInit)?

Database Debugging

If your database does not behave as expected:

- a) Verify that records scan correctly:
 - Check SCAN field is correct – periodic / passive / I/O interrupt
 - Check the forward link (FLNK) and the PP/CP attributes of the input/output data links
- b) Check that data input and output links are correct. Look out for links which **appear as CA links when they should be local database links: this usually means that the PV name in the link is wrong!**
- c) To get details of a record's fields use the IOC shell command:
dbpr <record name> 4 to report all the field values.

Database Debugging (Contd)



Database Debugging (Contd)

- d) If the SEVR field is Invalid and the STAT field is UDF the record was never processed (this might be OK).
- e) If records are still not being processed as expected:
 - a) Set the .TPRO field of a database record to 1
 - b) This will cause ALL records processed as part of this chain to print their name on the IOC console
 - c) Very useful to verify processing links in the database
 - d) Set it back to zero again afterwards!

```
epics>
epics> dbpf ajf67:counter.TPRO 1
DBR_UCHAR:      1          0x1
epics> scan1: Process ajf67:counter
scan1: Process ajf67:function
scan1: Process ajf67:counter
scan1: Process ajf67:counter
```

Channel Access Diagnostics from the IOC Shell

- **dbcar <record> <level>**

An IOC command which lists Channel Access links in the named record. <level> may be 0, 1 or 2.

Level 1 is useful, with record as ‘*’, this lists all the records where a CA link is *not* connected.

```
epics> dbcar * 1
CA links in all records

    ajf67:function.INPC --> ajf67:freqCal          (0, 0)

Total 1 CA link; 0 connected, 1 not connected.
  0 can't read, 0 can't write. (0 disconnects, 0 writes prohibited)

epics> □
```

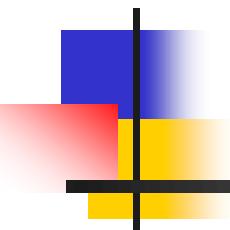
Channel Access Diagnostics from the IOC Shell

- **casr <level>**

An IOC command which lists the Channel Access Server status.

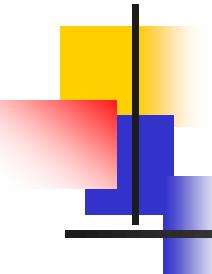
<level> may be 0, 1 or 2.

```
epics> casr 1
.
.
.
Channel Access Server V4.13
Connected circuits:
TCP 172.23.245.105:50510(pc0105.cs.diamond.ac.uk): User="ajf67", V4.13, 4 Channels, Priority=0
    Task Id=0x7faf040096f0, Socket FD=9
    Secs since last send  0.10, Secs since last receive  3.40
    Unprocessed request bytes=0, Undelivered response bytes=0
    State=up
epics> █
```



EPICS Channel Access Client Tools: Part 1

Andy Foster



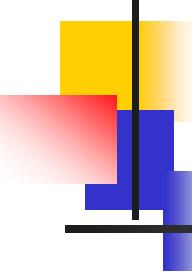
CA Command Line Utilities

Client side command line tools including the following text output commands:

- **caget, caput**
 - To get or put a channel value (no parameters displays help)
- **camonitor**
 - Outputs a value, with timestamp, for each monitored change on the named channel (or list of channels)
- **cainfo**
 - Outputs information, including the location and type of the channel

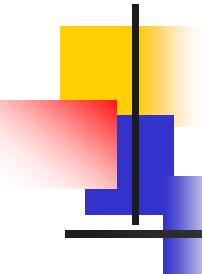
caget options/1

Option	Description							
-h	Print usage information							
	CA options:							
-w <sec>	Wait time, specifies CA timeout, default is 1.0 second(s)							
-c	Asynchronous get (use ca_get_callback and wait for completion)							
	Format and data type options:							
	Default output format is "name value"							
-t	Terse mode - print only value, without name							
-a	Wide mode "name timestamp value stat sevr" (read PVs as DBR_TIME_xxx)							
-n	Print DBF_ENUM values as number (default are enum strings)							
	Request specific dbr type; use string (DBR_ prefix may be omitted)							
	or number of one of the following types:							
-d <type>	DBR_STRING	0	DBR_STS_FLOAT	9	DBR_TIME_LONG	19	DBR_CTRL_SHORT	29
	DBR_INT	1	DBR_STS_ENUM	10	DBR_TIME_DOUBLE	20	DBR_CTRL_INT	29
	DBR_SHORT	1	DBR_STS_CHAR	11	DBR_GR_STRING	21	DBR_CTRL_FLOAT	30
	DBR_FLOAT	2	DBR_STS_LONG	12	DBR_GR_SHORT	22	DBR_CTRL_ENUM	31
	DBR_ENUM	3	DBR_STS_DOUBLE	13	DBR_GR_INT	22	DBR_CTRL_CHAR	32
	DBR_CHAR	4	DBR_TIME_STRING	14	DBR_GR_FLOAT	23	DBR_CTRL_LONG	33
	DBR_LONG	5	DBR_TIME_INT	15	DBR_GR_ENUM	24	DBR_CTRL_DOUBLE	34
	DBR_DOUBLE	6	DBR_TIME_SHORT	15	DBR_GR_CHAR	25	DBR_STSACK_STRING	37
	DBR_STS_STRING	7	DBR_TIME_FLOAT	16	DBR_GR_LONG	26	DBR_CLASS_NAME	38
	DBR_STS_SHORT	8	DBR_TIME_ENUM	17	DBR_GR_DOUBLE	27		
	DBR_STS_INT	8	DBR_TIME_CHAR	18	DBR_CTRL_STRING	28		



caget options/2

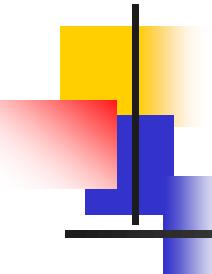
	Arrays:
	Value format: Print number of requested values, then list of values
Default:	Print all values
-# <count>	Print first <count> elements of an array
	Floating point type format:
Default:	Use %g format
-e <nr>	Use %e format, with <nr> digits after the decimal point
-f <nr>	Use %f format, with <nr> digits after the decimal point
-g <nr>	Use %g format, with <nr> digits after the decimal point
-s	Get value as string (may honour server-side precision)
	Integer number format:
Default:	Print as decimal number
-0x	Print as hex number
-0o	Print as octal number
-0b	Print as binary number



caget – Requesting data

```
[ajf67@pc0107 ~]$ caget -w 0.03 ajf67:amplitude
ajf67:amplitude          1
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$ caget -w 0.02 ajf67:amplitude
Channel connect timed out: 'ajf67:amplitude' not found.
[ajf67@pc0107 ~]$
```

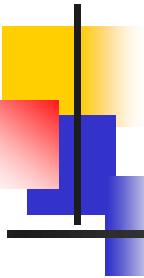
```
[ajf67@pc0107 ~]$ caget ajf67:amplitude
ajf67:amplitude          1
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$ caget -d DBR_DOUBLE ajf67:amplitude
ajf67:amplitude
  Native data type: DBF_DOUBLE
  Request type:    DBR_DOUBLE
  Element count:   1
  Value:           1
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$ caget -d DBR_STS_DOUBLE ajf67:amplitude
ajf67:amplitude
  Native data type: DBF_DOUBLE
  Request type:    DBR_STS_DOUBLE
  Element count:   1
  Value:           1
  Status:          NO_ALARM
  Severity:        NO_ALARM
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$ caget -d DBR_TIME_DOUBLE ajf67:amplitude
ajf67:amplitude
  Native data type: DBF_DOUBLE
  Request type:    DBR_TIME_DOUBLE
  Element count:   1
  Value:           1
  Timestamp:       2022-10-17 16:23:53.058821
  Status:          NO_ALARM
  Severity:        NO_ALARM
[ajf67@pc0107 ~]$
```



caget – Requesting data

```
[ajf67@pc0107 ~]$ caget -d DBR_GR_DOUBLE ajf67:amplitude
ajf67:amplitude
  Native data type: DBF_DOUBLE
  Request type:   DBR_GR_DOUBLE
  Element count:  1
  Value:          1
  Status:         NO_ALARM
  Severity:       NO_ALARM
  Units:          mm
  Precision:      0
  Lo disp limit: -1000
  Hi disp limit: 1000
  Lo alarm limit: nan
  Lo warn limit:  nan
  Hi warn limit:  nan
  Hi alarm limit: nan
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$ caget -d DBR_CTRL_DOUBLE ajf67:amplitude
ajf67:amplitude
  Native data type: DBF_DOUBLE
  Request type:   DBR_CTRL_DOUBLE
  Element count:  1
  Value:          1
  Status:         NO_ALARM
  Severity:       NO_ALARM
  Units:          mm
  Precision:      0
  Lo disp limit: -1000
  Hi disp limit: 1000
  Lo alarm limit: nan
  Lo warn limit:  nan
  Hi warn limit:  nan
  Hi alarm limit: nan
  Lo ctrl limit: -1001
  Hi ctrl limit: 1001
[ajf67@pc0107 ~]$ █
```

```
record(ao, "$(user):amplitude") {
    field(PINI, "YES")
    field(DRVH, "1001")
    field(DRVL, "-1001")
    field(HOPR, "1000")
    field(LOPR, "-1000")
    field(EGU, "mm")
    field(VAL, "1.000000")
}
```

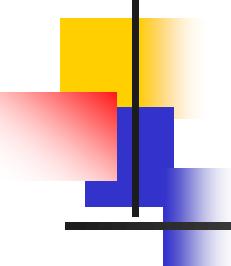


caget – Enums & Waveforms

```

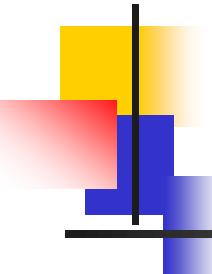
record(mbbo, "$(user):frequency") {
    field(SCAN, "Passive")
    field(PINI, "YES")
    field(ZRVL, "10")
    field(ONVL, "5")
    field(TWVL, "2")
    field(THVL, "1")
    field(ZRST, "1 Hz")
    field(ONST, "0.5 Hz")
    field(TWST, "0.2 Hz")
    field(THST, "0.1 Hz")
    field(DTYP, "Raw Soft Channel")
}

```



caput options

Option	Description
-h	Print usage information
	CA options:
-w <sec>	Wait time, specifies CA timeout, default is 1.0 second(s)
-c	Asynchronous put (use ca_put_callback and wait for completion)
	Format options:
-t	Terse mode - print only successfully written value, without name
	Enum Format:
	Auto - try value as ENUM string, then as index number
-n	Force interpretation of values as numbers
-s	Force interpretation of values as strings
	Arrays:
-a	Put array data
	Value format: Print number of requested values, then list of values



camonitor options

Usage: camonitor [options] <PV name> ...

-h: Help

Channel Access options:

-w <sec> : Wait time, specifies CA timeout, default is 1s
-m <mask> : Specify CA event mask to use,
with <mask> being any combination of
'v' (value), 'a' (alarm), 'l' (log or archiver).
Default: va

Timestamps:

Default: Print absolute timestamps (as reported by server)
-t <key>: Specify timestamp source(s) and type, with
<key> containing:
's' = CA server (remote) timestamps
'c' = CA client (local) timestamps (shown in ')'s)
'n' = no timestamps
'r' = relative timestamps (time since start of program)
'i' = incremental timestamps (time since last update)
'I' = incremental timestamps (time elapsed since last
update, by channel)

Enumeration format:

-n: Print DBF_ENUM values as number (default are enum
string values)

Arrays:

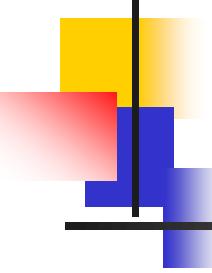
Default : Print all values
-# <count>: Print first <count> elements of an array

Floating point type format:

Default : Use %g format
-e <nr> : Use %e format, with a precision of <nr> digits
-f <nr> : Use %f format, with a precision of <nr> digits
-g <nr> : Use %g format, with a precision of <nr> digits
-s : Get value as string

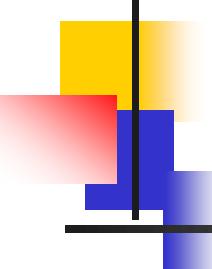
Integer number format:

Default: Print as decimal number
-0x: Print as hex number
-0o: Print as octal number
-0b: Print as binary number



camonitor output

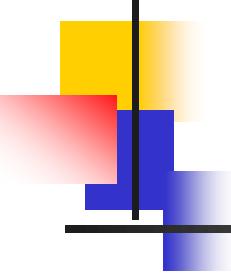
```
[ajf67@pc0107 ~]$ camonitor ajf67:function
ajf67:function          2022-10-17 17:45:54.858927 -0.951056
ajf67:function          2022-10-17 17:45:54.958827 -0.587785
ajf67:function          2022-10-17 17:45:55.058731 0
ajf67:function          2022-10-17 17:45:55.158817 0.587785
ajf67:function          2022-10-17 17:45:55.258922 0.951057
ajf67:function          2022-10-17 17:45:55.358770 0.951057
ajf67:function          2022-10-17 17:45:55.458891 0.587785
ajf67:function          2022-10-17 17:45:55.558810 -4.64102e-08
ajf67:function          2022-10-17 17:45:55.658724 -0.587785
ajf67:function          2022-10-17 17:45:55.758835 -0.951057
ajf67:function          2022-10-17 17:45:55.858924 -0.951056
ajf67:function          2022-10-17 17:45:55.958762 -0.587785
ajf67:function          2022-10-17 17:45:56.058772 0
ajf67:function          2022-10-17 17:45:56.158830 0.587785
ajf67:function          2022-10-17 17:45:56.258887 0.951057
ajf67:function          2022-10-17 17:45:56.358833 0.951057
ajf67:function          2022-10-17 17:45:56.458944 0.587785
ajf67:function          2022-10-17 17:45:56.558762 -4.64102e-08
ajf67:function          2022-10-17 17:45:56.658935 -0.587785
ajf67:function          2022-10-17 17:45:56.758831 -0.951057
ajf67:function          2022-10-17 17:45:56.858895 -0.951056
ajf67:function          2022-10-17 17:45:56.958837 -0.587785
ajf67:function          2022-10-17 17:45:57.058766 0
ajf67:function          2022-10-17 17:45:57.158787 0.587785
ajf67:function          2022-10-17 17:45:57.258939 0.951057
ajf67:function          2022-10-17 17:45:57.358775 0.951057
ajf67:function          2022-10-17 17:45:57.458919 0.587785
ajf67:function          2022-10-17 17:45:57.558711 -4.64102e-08
ajf67:function          2022-10-17 17:45:57.658942 -0.587785
ajf67:function          2022-10-17 17:45:57.758806 -0.951057
ajf67:function          2022-10-17 17:45:57.858913 -0.951056
ajf67:function          2022-10-17 17:45:57.958834 -0.587785
ajf67:function          2022-10-17 17:45:58.058735 0
ajf67:function          2022-10-17 17:45:58.158843 0.587785
ajf67:function          2022-10-17 17:45:58.258938 0.951057
ajf67:function          2022-10-17 17:45:58.358804 0.951057
^C
```



cainfo options

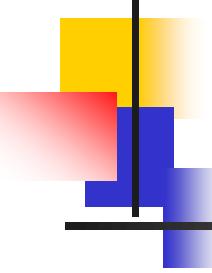
Option	Description
-h	Print usage information
	CA options:
-w <sec>	Wait time, specifies CA timeout, default is 1.0 second(s)
-s <level>	Call ca_client_status with the specified interest level

```
[ajf67@pc0107 ~]$ cainfo ajf67:frequency
ajf67:frequency
  State:          connected
  Host:          172.23.245.107:5064
  Access:         read, write
  Native data type: DBF_ENUM
  Request type:   DBR_ENUM
  Element count:  1
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$ cainfo ajf67:waveform
ajf67:waveform
  State:          connected
  Host:          172.23.245.107:5064
  Access:         read, write
  Native data type: DBF_CHAR
  Request type:   DBR_CHAR
  Element count:  256
[ajf67@pc0107 ~]$
[ajf67@pc0107 ~]$
```



capr.pl command

- A Perl script called `capr.pl` is available for displaying details of all fields in a record
- It can also list available record types and fields in a record.



capr.pl options

Usage: capr.pl -h

```
capr.pl [-d file.dbd] [-w seconds] -r  
capr.pl [-d file.dbd] [-w seconds] -f record_type  
capr.pl [-d file.dbd] [-w seconds] record_name [interest]
```

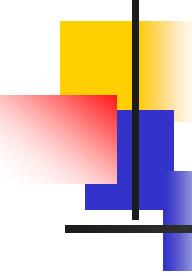
Description:

Attempts to perform a "dbpr" record print via channel access for *record_name* at an interest level which defaults to level 0.

The -r or -f options cause it to print record type or field lists.

Options:

- | | |
|-----------------|---|
| -h | Prints this help message |
| -r | Lists all record types in the dbd file |
| -f record_type: | Lists the interest level, data type and base type for all fields in the given record type |
| -d file.dbd: | Specify this dbd file containing record type definitions
This can be set using the EPICS_CAPR_DB_FILE environment variable, currently
/dls_sw/epics/R3.14.12.7/base/dbd/softIoc.dbd |
| -w seconds: | Wait time, specifies CA timeout, default 1 second |



capr.pl -r

```
[ajf67@pc0107 ~]$ capr.pl -r
Using /dls_sw/epics/R3.14.12.7/base/bin/linux-x86_64/.../dbd/softLoc.dbd

Record types found:
aSub
aai
ao
ai
ao
bi
bo
calc
calcout
compress
dfanout
event
fanout
longin
longout
mbbi
mbbiDirect
mbbo
mbboDirect
permissive
sel
seq
state
stringin
stringout
sub
subArray
waveform
[ajf67@pc0107 ~]$
```

capr.pl -f mbbo

```
[ajf67@pc0107 ~]$ capr.pl -f mbbo
Using /dls_sw/epics/R3.14.12.7/base/bin/linux-x86_64/.../dbd/softIoc.dbd

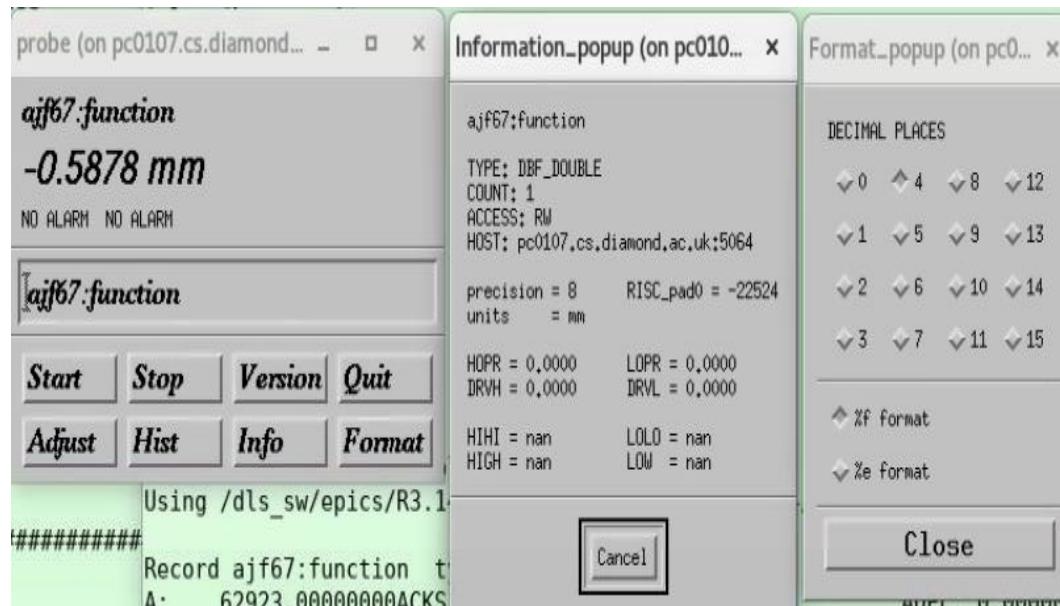
Record type - mbbo
ACKS interest = 2      type = DBF_MENU          base = DECIMAL
ACKT interest = 2      type = DBF_MENU          base = DECIMAL
ASG interest = 0       type = DBF_STRING        base = DECIMAL
ASP interest = 4       type = DBF_NOACCESS      base = DECIMAL
BKPT interest = 1      type = DBF_NOACCESS      base = DECIMAL
COSV interest = 1      type = DBF_MENU          base = DECIMAL
DESC interest = 0      type = DBF_STRING        base = DECIMAL
DISA interest = 0      type = DBF_SHORT         base = DECIMAL
DISP interest = 0      type = DBF_UCHAR         base = DECIMAL
DISS interest = 1      type = DBF_MENU          base = DECIMAL
DISV interest = 0      type = DBF_SHORT         base = DECIMAL
DOL interest = 1      type = DBF_INLINK        base = DECIMAL
DPVT interest = 4      type = DBF_NOACCESS      base = DECIMAL
DSET interest = 4      type = DBF_NOACCESS      base = DECIMAL
DTYP interest = 1      type = DBF_DEVICE        base = DECIMAL
EIST interest = 1      type = DBF_STRING        base = DECIMAL
EISV interest = 1      type = DBF_MENU          base = DECIMAL
EIVL interest = 1      type = DBF ULONG          base = HEX
ELST interest = 1      type = DBF_STRING        base = DECIMAL
ELSV interest = 1      type = DBF_MENU          base = DECIMAL
ELVL interest = 1      type = DBF ULONG          base = HEX
EVNT interest = 1      type = DBF_SHORT         base = DECIMAL
FFST interest = 1      type = DBF_STRING        base = DECIMAL
FFSV interest = 1      type = DBF_MENU          base = DECIMAL
FFVL interest = 1      type = DBF ULONG          base = HEX
FLNK interest = 1      type = DBF_FWDLINK       base = DECIMAL
FRST interest = 1      type = DBF_STRING        base = DECIMAL
FRSV interest = 1      type = DBF_MENU          base = DECIMAL
FSWV interest = 1      type = DBF ULONG          base = HEX
```

capr.pl ajf67:function 3

```
[ajf67@pc0107 ~]$ capr.pl ajf67:function 3
Using /dls_sw/epics/R3.14.12.7/base/bin/linux-x86_64/.../dbd/softIoc.dbd

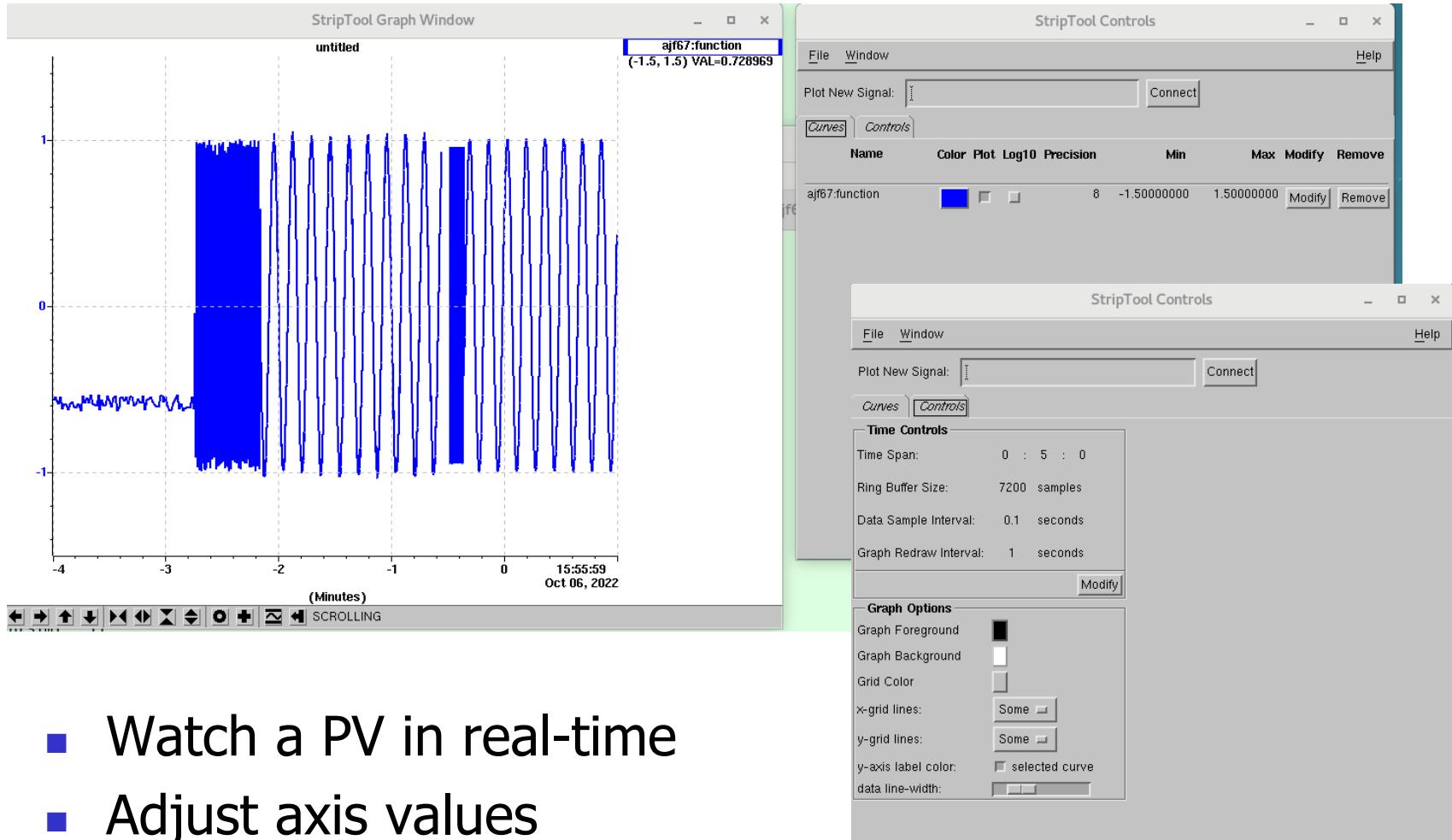
Record ajf67:function type calc
A: 62923.00000000ACKS: NO_ALARM      ACKT: YES          ADEL: 0.00000000
ALST: 0.95105651 ASG:                 B: 10.00000000   C: 0.00000000
CALC: C+(D*SIN(B*(A%(100/B))*2*0.031415927))+ D: 1.00000000
DESC:                   DISA: 0          DISP: 0          DISS: NO_ALARM
DISV: 1                  E: 0.00000000   EGU: mm          EVNT: 0
F: 0.00000000  FLNK: 0          G: 0.00000000   H: 0.00000000
HHSV: NO_ALARM    HIGH: 0.00000000  HIHI: 0.00000000 HOPR: 0.00000000
HSV: NO_ALARM     HYST: 0.00000000  I: 0.00000000
INPA: ajf67:counter NPP NMS        INPB: ajf67:frequency.RVAL NPP NMS
INPC: ajf67:offset.VAL NPP NMS    INPD: ajf67:amplitude NPP NMS
INPE: ajf67:noise NPP NMS       INPF:             INPG:
INPH:                   INPI:          INPJ:             INPK:
INPL:                   J: 0.00000000  K: 0.00000000  L: 0.00000000
LA: 62923.00000000LALM: 0.95105651 LB: 10.00000000 LC: 0.00000000
LCNT: 0                  LD: 1.00000000  LE: 0.00000000 LF: 0.00000000
LG: 0.00000000  LH: 0.00000000  LI: 0.00000000 LJ: 0.00000000
LK: 0.00000000  LL: 0.00000000  LLSV: NO_ALARM   LOLO: 0.00000000
LOPR: 0.00000000  LOW: 0.00000000  LSV: NO_ALARM   MDEL: 0.00000000
MLST: 0.95105651 NAME: ajf67:functionNSEV: NO_ALARM  NSTA: NO_ALARM
PACT: 0                  PHAS: 0         PINI: NO        PREC: 8
PRIO: LOW                PROC: 0         PUTF: 0         RPRO: 0
SCAN: Passive            SDIS:          SEVR: NO_ALARM  STAT: NO_ALARM
TPRO: 0                  TSE: 0          TSEL:           UDF: 0
VAL: 0.95105651
[ajf67@pc0107 ~]$
```

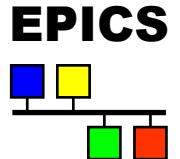
Probe



- Enter name of record to be displayed
- Start/Stop continuous display of value
- Adjust to set value, if writable
- Information and Format windows

StripTool



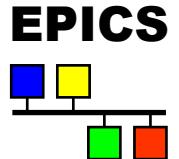


Channel Access

Andy Foster
Observatory Sciences Limited



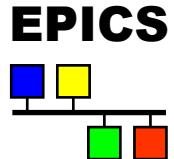
Outline



- What is Channel Access?
- Channel Naming Convention
- Features of Channel Access
- Asynchronous nature of Channel Access
- Client Library Functions
 - Create context
 - Create channel
 - Put
 - Get
 - Asynchronous Events
 - Exceptions
 - Flushing the send buffer
 - Error reporting
- Channel Access configuration
- Connection management
- Examples of Channel Access client code
- Documentation



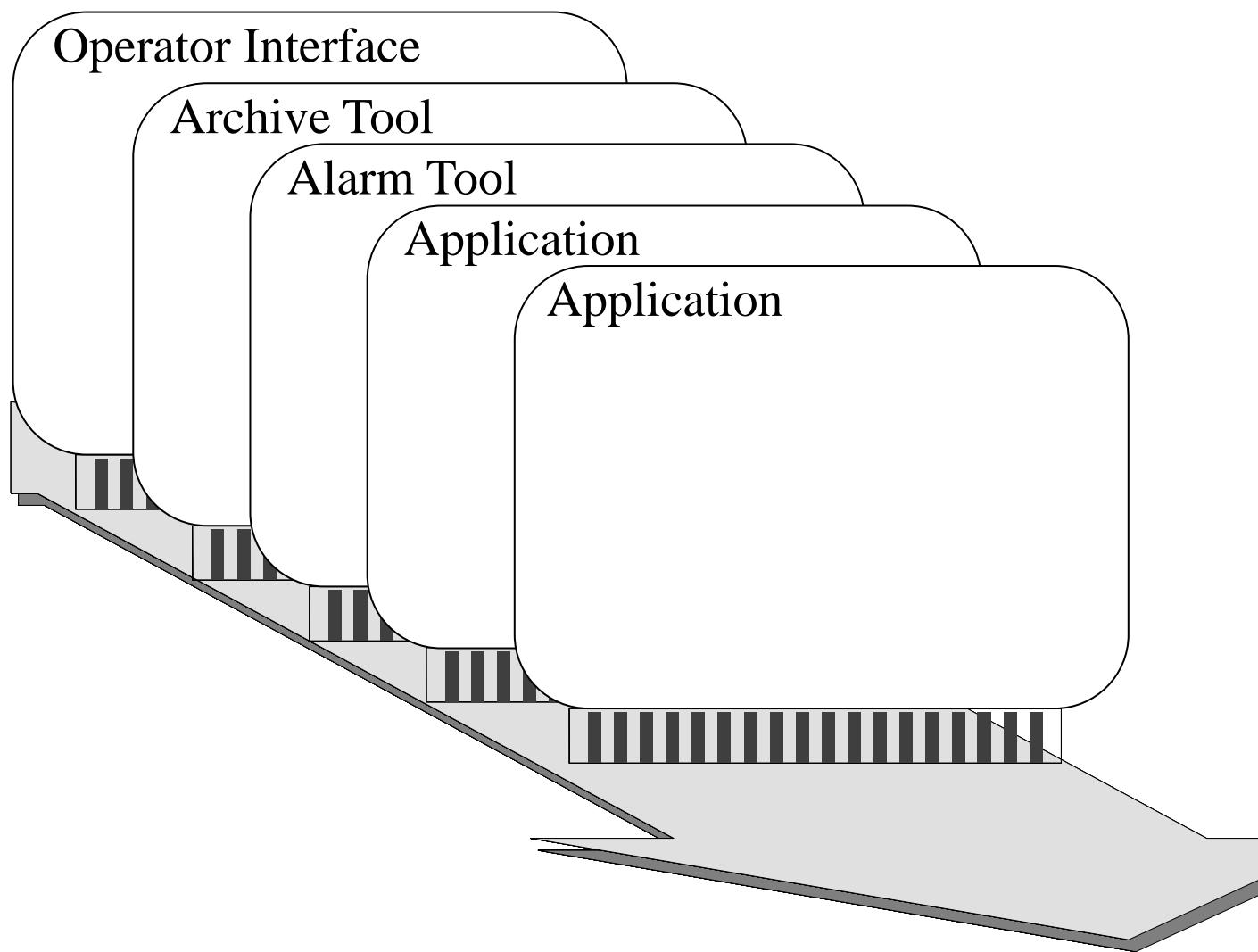
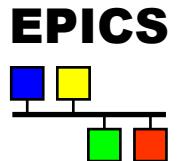
What is Channel Access?



- Channel Access (CA) is the EPICS communication protocol.
- Based on a client-server architecture.
- Clients are written using the CA client library.
- The server is part of iocCore.
- Note that an IOC can be both a client and a server!
- Clients make requests to the servers across the network.
- A “Channel” is the communication path to a field within a record (*process variable*) in an IOC.
- Integrates software modules into the control system i.e. Alarm Handler, Operator Interface.

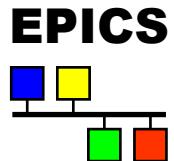


The “EPICS Software Bus”





Channel Naming Convention

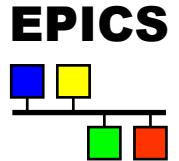


- CA requires that channels have a name.
- The IOC database requires names of the form:
 - <record name>[.<field name>]
 - [BL24I-MO-IOC-01:OMEGA.LOPR](#)"
 - [BL24I-MO-IOC-01:OMEGA](#)
 - If the field name is omitted [.VAL](#) is assumed.
- For large projects, record names usually follow a record naming convention.
- Record field names and purposes are record type specific
- A list of the field names available for each record can be found in the:

EPICS Record Reference Manual.



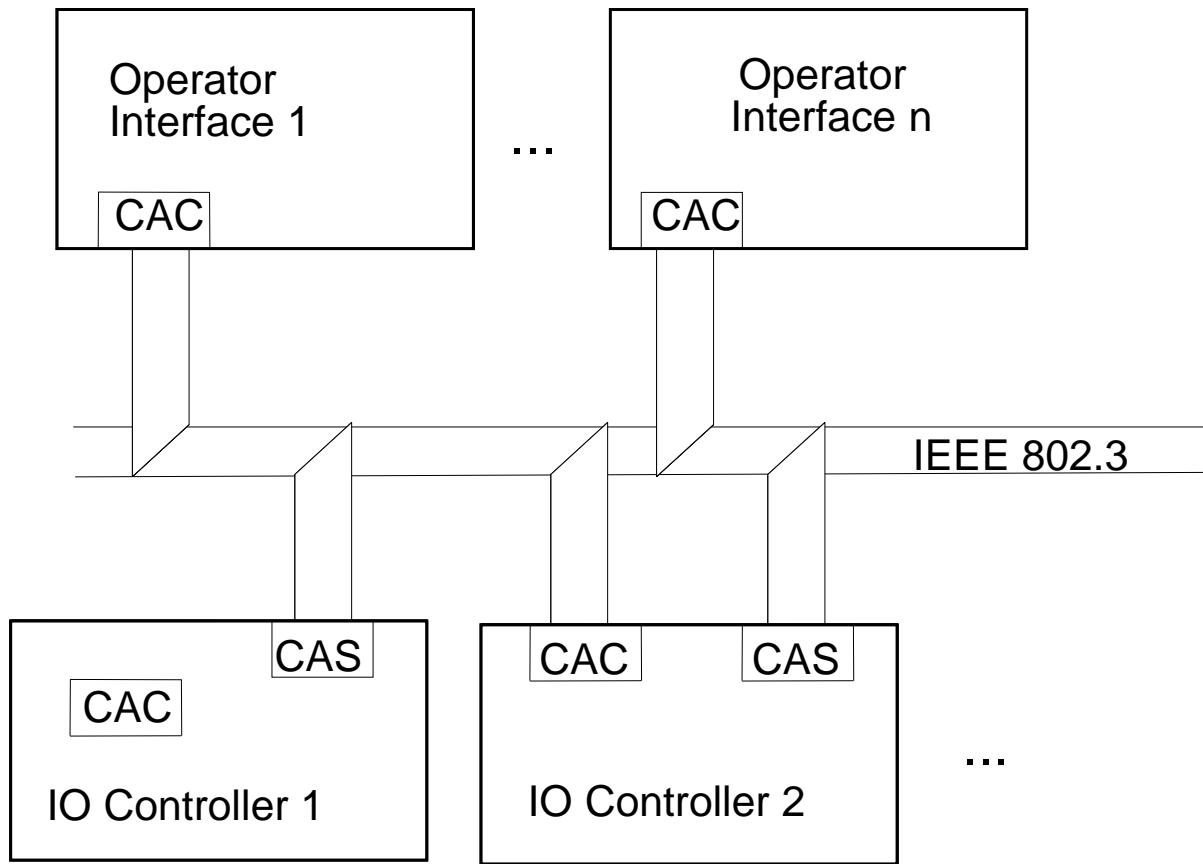
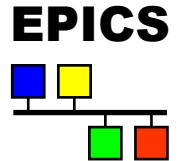
Features of Channel Access



- The client library is easy to use (see later)
- It is operating system transparent
- It provides network transparency (i.e. access to remote and local channels is identical)
- Handles data conversion between different CPU architectures
- Asynchronous and efficient...
 - Small packet sizes
 - small headers
 - combined operations on the network



Channel Access Network Architecture

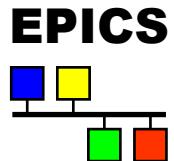


CAS Channel Access Server

CAC Channel Access Client



Asynchronous Nature of Channel Access

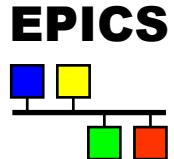


- CA does not wait to gain access to the network prior to returning from each client library call.
 - Applications continue to run on the local processor while several operations are completing on remote processors.
- CA library calls are buffered and sent when either:
 - the buffer fills (default size: 16384 bytes)
 - a special CA library call is made
 - Because combined operations are more efficient when sharing a common resource such as the network.
- CA model fits well with data fetched from a remote machine, which is often asynchronous
- All operations are guaranteed to be executed in the order requested.



Client Library Functions

Create CA Context

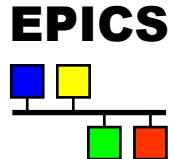


- **ca_context_create(**
enum ca_preemptive_callback_select SELECT)
- Should be called once from each thread prior to making any other CA calls.
- SELECT argument: non-preemptive mode:
 - a user's callback functions will only be called when the thread which made this call is inside a CA library function.
 - CA client library always waits for the current callback to complete before invoking a new one.
 - This means the user does not have to worry about multiple, simultaneous, accesses to data structures.
 - Generally, single-threaded programs
- SELECT argument: pre-emptive mode:
 - a user's callback functions can be called at anytime
 - user is responsible for mutex locking of private data structures.
 - Generally, multi-threaded programs
- Registers this client with the CA repeater.
 - Process which fans out UDP server beacons received on the CA UDP port to all CA clients running on the machine.
 - Solves the problem of the O/S only allowing one process to have access to the port.



Client Library Functions

Create Channel

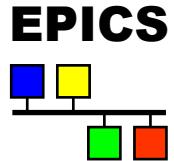


- **ca_create_channel(“name”, connectFunc,
&user, priority,
&channelID)**
- Causes a UDP broadcast to be sent to all CA servers on the client's local subnet.
- Only a server which recognizes “name” will respond to the client
 - If identical record names exist in two IOCs, the first to reply “wins” the connection
- The client library then opens a TCP connection with the server and establishes a “virtual circuit” to access the channel
 - Note that there is only one TCP connection between any particular client and a server. All data about all channels (PV's) on that IOC pass through this connection.
 - Connections between individual channels in the IOC and the client are referred to as “virtual circuits”.



Client Library Functions

Create Channel (contd.)

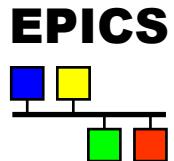


- *connectFunc* is an optional user function to call when the connection state changes.
- *user* is an optional pointer to user data which will be associated with this channel.
- *priority* Not implemented. Was added to set the priority level for servicing requests within the server (range is 0 - 99 with 0 the lowest).
- *channelID* is an identifier, returned from the client library, for this channel. It will be used to communicate with this channel in all future CA calls made by the users application.



Client Library Functions

Put

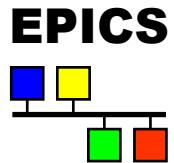


- There are many CA PUT functions:
 - `ca_put(field_type, channelId, &val)`
 - `ca_put_callback(field_type, channelId, &val, usrFunc, usrArg)`
- *field_type*:
 - All channels have a “native” data storage type in the IOC e.g. integer, floating point, string, enumerated type
 - *field_type* is the data format we will use to write to the channel. Often referred to as the external data type.
 - The external data type can be different from the native type if conversion is possible.
- Conversion between types is done in the server
- Ask for native types for better performance at the real-time end!
- All use “channel ID” from *Create* to communicate with record field
- *usrFunc* is called after all records in the processing chain in the database have been processed.



Client Library Functions

Get

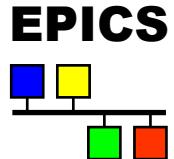


- There are many CA GET functions:
 - `ca_get(field_type, channelId, &val)`
 - `ca_get_callback(field_type, channelId, usrFunc, usrArg)`
- Arguments similar to the PUT functions.
- Notice that `ca_get_callback` does not get the value directly. It is available to the user defined function (see example).



Client Library Functions

Asynchronous Events

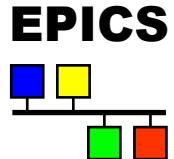


- How can we write a client which will react to asynchronous events in the database?
- Use:
ca_create_subscription(
 field_type, count, channelId,
 mask, usrFunc, usrArg, &evid)
- *usrFunc* will be called when the “events” we have subscribed to, occur in the database. We define these “events” by the *mask* parameter.
- *mask* will be a logical OR of:
 - DBE_VALUE: Channel value changes by more than the monitor deadband.
 - DBE_LOG: Channel value changes by more than the archival deadband.
 - DBE_ALARM: Alarm status or severity changes.
- *evid* is a returned pointer to this event. It can be used as an argument to **ca_clear_event** to cancel the subscription to these event(s).
- Events are placed in a queue and handled in the order that they occurred.
- For analogue channels, the rate at which events occur should be minimized so that we do not saturate the network. This rate is managed by adjusting the channel’s dead band and scan rate.



Client Library Functions

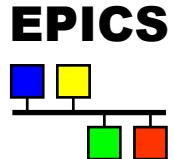
Exceptions



- Since the CA client library does not wait to gain access to the network before returning from each call, it does not know if the operation was successful in the server.
 - error codes returned from CA library calls check the validity of the request NOT the success of the operation.
- Status of unsuccessful operations are returned from the server to the client's exception handler.
- The default exception handler prints a message for each unsuccessful operation and aborts the client if the condition is severe.
- We can install our own exception handler by using:
ca_add_exception_event
(pCallback userFunc, void *usrArg);
where:
typedef void (*pCallback)
(struct exception_handler_args hArgs);
- Operations which fail in the server are nearly always caused by programming errors e.g. trying to convert between two incompatible types (e.g. string to double).



Flushing the Send Buffer

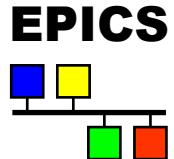


- Vital for the CA client to work!!
- All CA client calls are buffered until either the send buffer is full or one of these is explicitly called:
 - **ca_pend_io(timeout)**
Flushes the send buffer and waits until the shorter of:
 - All outstanding calls to ca_get complete
 - The timeout expires

(Note a timeout of 0 means wait forever)
 - **ca_pend_event(timeout)**
Flushes the send buffer and always waits “timeout” seconds for asynchronous events
(timeout of 0 means wait forever)



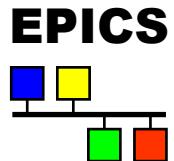
Error Reporting



- For portability reasons CA functions do not return status following the UNIX convention.
- Do not test return status against “0”.
- Useful macro for testing return status:
 - SEVCHK(status, “user string”);



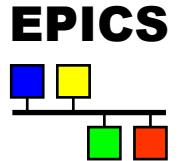
Channel Access Configuration



- CA clients and servers can be configured by setting environment variables
- On Unix/Linux:
 - csh, tcsh – `setenv VARNAME value`
 - sh, bash, ksh – `export VARNAME=value`
 - `printenv` displays all variables from any shell
- On vxWorks:
 - `putenv "VARNAME=value"`
 - `envShow` displays all variable values
- EPICS soft IOC shell iocsh:
 - `epicsEnvSet VARNAME value`
- Environment variables are inherited when you start a new program, not afterwards
 - Unix: Set the variables, then start the client
 - vxWorks: Set variables in the startup script
- Default values for a site are set, system-wide, in:
`<epics>/base/config/CONFIG_ENV` and
`<epics>/base/config/CONFIG_SITE_ENV`



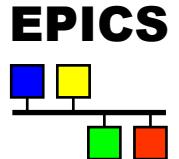
Connection Management



- Network connections are inherently transient.
- A channel's connection state is either not found, connected, or disconnected.
- CA allows you to specify a handler to be run when a channel's connection state changes.
- Connection requires:
 - a CA server for the channel
 - a valid network path to the server.
- CA automatically restores connection upon notification from the server. This is a very strong feature of CA!



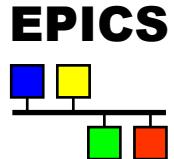
Connection Health



- By default, CA servers send an:
 - “I’m still here” UDP broadcast beacon every 15 seconds.
 - This is known as the **Server Beacon Period**.
- If a server is quiet for 30 seconds (the **Client Connection Timeout**), any connected clients will
 - send it an “echo” packet (not broadcast)
 - allow 5 seconds for it to reply
 - mark all channels to this server disconnected
- Potential problems:
 - Busy sites may want to reduce broadcast rates
 - Clients take 35 seconds to recognize when a server has died



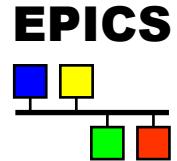
Configuring Connection Health



- How to change the **Server Beacon Period?**
 - In the EPICS startup script:
 - `epicsEnvSet EPICS_CA_BEACON_PERIOD 30.0`
 - Default value is 15.0 seconds
- How to change the **Client Connection Timeout?**
 - In a bash terminal, before running client:
 - `export EPICS_CA_CONN_TMO=60.0`
 - Default value is 30.0 seconds
 - This value determines how long a client takes to notice that a server has died (+5 seconds)
- The client connection timeout must be longer than the server beacon period.
- Recommendation: At least twice as long.



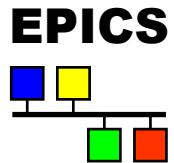
Configuring Name Resolution



- How to disable all broadcasts?
 - `export EPICS_CA_AUTO_ADDR_LIST=NO`
 - Default value = YES
 - Remember: IOCs can also be clients, so generate broadcasts
- How to find channels without broadcast?
 - Use: `EPICS_CA_ADDR_LIST`
 - List of IP addresses, separated by spaces
- `export EPICS_CA_ADDR_LIST="172.23.124.71 172.23.124.72"`
 - This list is used in addition to broadcasts if these are enabled
 - A port number may optionally be specified after a trailing colon.
- How to search other subnets as well?
 - Use a **broadcast address** in `EPICS_CA_ADDR_LIST`
`export EPICS_CA_ADDR_LIST="172.23.124.255"`
 - Note: Some routers will not pass broadcast addresses!



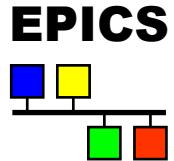
Configuring Port Numbers



- Channel Access uses two IP port numbers for its communication
 - EPICS_CA_SERVER_PORT
 - Default is 5064
 - UDP used for name resolution
 - TCP used for channel requests
 - EPICS_CA_REPEATERS_PORT
 - Default is 5065
 - UDP used for receiving server beacons
 - Both should be > 5000, check with sys-admins
- The settings for a server and all its clients must be the same
- Using different port numbers can allow independent projects to share a subnet without any danger of CA name clashes
 - Can also be used for application testing
 - No interaction is possible between projects



Configuring Maximum Array Size

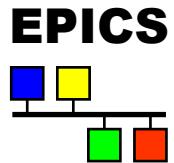


- Channel Access has a default maximum array size of 16384 bytes. This is the size of the largest array that can be passed through CA in a single transfer.
- This value may be changed using the environment variable
`EPICS_CA_MAX_ARRAY_BYTES`
- A value > 16384 bytes will allocate more buffer space.
- The settings for a server and all its clients must be the same



Example Client

caPut.c



```
#include <stdio.h>
#include <stdlib.h>
#include <cadef.h> /* Structures and data types used by CA */

int main( int argc, char *argv[] )
{
    int      status;
    chid    channelId;
    double  val;

    val = atof( argv[1] );

    status = ca_context_create( ca_disable_preemptive_callback );
    SEVCHK(status, " ");

    status = ca_create_channel( "test:ai", 0, 0, 0, &channelId );
    SEVCHK(status, " ");

    status = ca_pend_io(0.0);
    SEVCHK(status, " ");

    status = ca_put( DBR_DOUBLE, channelId, &val );
    SEVCHK(status, " ");

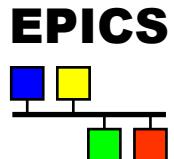
    status = ca_pend_io(0.0);
    SEVCHK(status, " ");

    return(0);
}
```



Example Client

caGet.c (main)



```
int main( int argc, char *argv[] )
{
    int      status;
    chid    channelId;
    info_t  info;

    info.project = "Diamond";
    info.numLoc = 167;
    info.current = 3.45;

    status = ca_context_create( ca_disable_preemptive_callback );
    SEVCHK(status, " ");

    status = ca_create_channel( "test:ai", 0, 0, 0, &channelId );
    SEVCHK(status, " ");

    status = ca_pend_io(0.0);
    SEVCHK(status, " ");

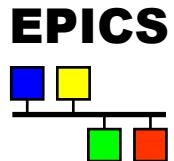
    status = ca_get_callback( DBR_DOUBLE, channelId,
                            (void (*)())usrFunc, &info );
    SEVCHK(status, " ");

    status = ca_pend_event(0.1); /* This is different! */
    SEVCHK(status, " ");
    return(0);
}
```



Example Client

caGet.c (contd.)



```
#include <stdio.h>
#include <stdlib.h>
#include <cadef.h>

typedef struct info
{
    char *project;
    int numLoc;
    double current;
} info_t;

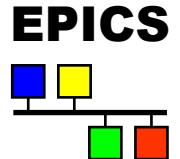
void usrFunc( struct event_handler_args args ) /* cadef.h */
{
    info_t *t = (info_t *)args.usr;

    if( args.status == ECA_NORMAL )
    {
        printf("UsrFunc called: Value      = %f\n", *(double *)args.dbr);
        printf("User Argument Name     = %s\n", t->project);
        printf("User Argument NumLoc = %d\n", t->numLoc);
        printf("User Argument Current = %f\n", t->current);
        printf("Channel Name          = %s\n", ca_name(args.chid) );
        printf("Number of Elements    = %d\n", ca_element_count(args.chid) );
        printf("Host Name              = %s\n", ca_host_name(args.chid) );
    }
}
```



Example Client

caMonitor.c (main)



```
int main( int argc, char *argv[] )
{
    int      status;
    chid    channelId;

    status = ca_context_create( ca_disable_preemptive_callback );
    SEVCHK(status, " ");

    status = ca_create_channel( "test:ai", (void (*)())connectFunc,
                               0, 0, &channelId );
    SEVCHK(status, " ");

    status = ca_pend_io(0.0);
    SEVCHK(status, " ");

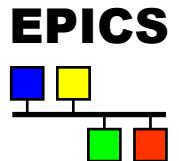
    status = ca_create_subscription( DBR_DOUBLE, 0, channelId,
                                    DBE_VALUE | DBE_LOG | DBE_ALARM,
                                    (void (*)())monitorFunc, NULL, NULL );
    SEVCHK(status, " ");

    status = ca_pend_event(0.0); /* Wait forever */
    SEVCHK(status, " ");
    return(0);
}
```



Example Client

caMonitor.c (contd.)



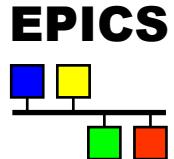
```
#include <stdio.h>
#include <stdlib.h>
#include <cadef.h>

void connectFunc( struct connection_handler_args args )
{
    if( ca_state(args.chid) != cs_conn )
        printf("%s has just Disconnected\n",
               ca_name(args.chid));
    else
        printf("%s has just Connected\n", ca_name(args.chid));
}

void monitorFunc( struct event_handler_args args )
{
    printf("Monitor on %s, new value = %f\n",
          ca_name(args.chid), *(double *)args.dbr);
}
```



Compound Data Types

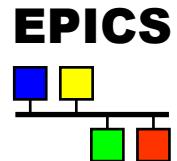


- Compound data types contain a channel value combined with additional status or configuration information.
 - Value
 - Alarm Status
 - Alarm Severity
 - Alarm Limits
 - Precision
 - Engineering units
 - Time Stamp
 - Display Limits
- Compound types involve the database record as a whole.
- Compound types can only be used with gets and monitors.
- Data types are described in:
\$EPICS_BASE/src/ca/db_access.h. (DBR_XXXX)



Example Client

caGetCompound.c (main)



```
int main( int argc, char *argv[] )
{
    int                     status;
    chid                   channelId;
    struct dbr_ctrl_double data;
    struct dbr_time_double tdata;
    char                   timeString[64];

    status = ca_context_create( ca_disable_preemptive_callback );
    SEVCHK(status, " ");

    status = ca_create_channel( "test:ai", 0, 0, 0, &channelId );
    SEVCHK(status, " ");

    status = ca_pend_io(0.0);
    SEVCHK(status, " ");

    status = ca_get( DBR_CTRL_DOUBLE, channelId, &data );
    SEVCHK(status, " ");

    status = ca_get( DBR_TIME_DOUBLE, channelId, &tdata );
    SEVCHK(status, " ");

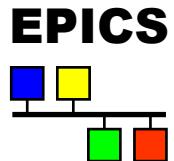
    status = ca_pend_io(0.0);
    SEVCHK(status, " ");
    epicsTimeToStrftime( timeString, sizeof(timeString),
                         "%a %b %d %Y %H:%M:%S.%f", &tdata.stamp );
    printResults( data, timeString );
    return(0);
}
```



Example Client

caGetCompound.c

(contd.)

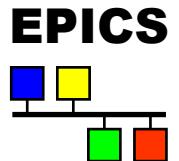


```
#include <stdio.h>
#include <stdlib.h>
#include <cadef.h>

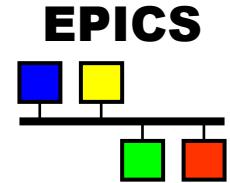
void printResults( struct dbr_ctrl_double data, char *timeString )
{
    printf("Channel Value      = %f\n", data.value);
    printf("Alarm Status       = %d\n", data.status); /* see alarm.h */
    printf("Alarm Severity     = %d\n", data.severity); /* see alarm.h */
    printf("Precision          = %d\n", data.precision);
    printf("Engineering Units  = %s\n", data.units);
    printf("Upper Display Limit = %d\n",
           data.upper_disp_limit);
    printf("Lower Display Limit = %d\n",
           data.lower_disp_limit);
    printf("Upper Alarm Limit  = %d\n",
           data.upper_alarm_limit);
    printf("Lower Alarm Limit  = %d\n",
           data.lower_alarm_limit);
    printf("Upper Warning Limit = %d\n",
           data.upper_warning_limit);
    printf("Lower Warning Limit = %d\n",
           data.lower_warning_limit);
    printf("Last Processed on: %s\n", timeString);
}
```



Documentation



- Documentation:
 - EPICS 3.14 Channel Access Reference Manual
[\(http://www.aps.anl.gov/epics/base/R3-14/10-docs/CAref.html\)](http://www.aps.anl.gov/epics/base/R3-14/10-docs/CAref.html)
 - Channel Access has now been replaced with *pvAccess* in EPICS 7, so there will not be any major new releases of CA.

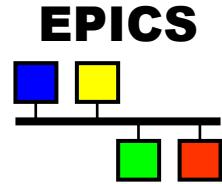


Introduction to Python scripting using caTools and aioca

Rebecca Williams
Observatory Sciences Ltd



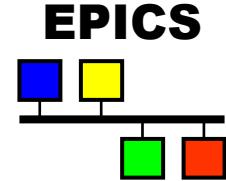
Python *Introduction*



- Python is an object-oriented scripting language developed by Guido van Rossum. With clear yet powerful syntax. Python is suitable for rapid application development and has become one of the most popular scripting languages.
- Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible.
- An important feature of Python is the "module". Functions and classes in the module can be imported into a Python program as a library. Modules can be either written in Python, C or other languages.



Python Introduction

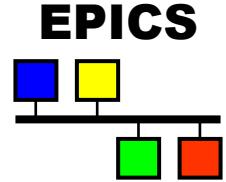


- The module structure allows the developer of a Python program to rewrite a module as an efficient C language module without affecting client applications.
- As of January 2016, the public repository of third-party software for Python contains more than 72,000 packages offering a wide range of functionality.
- Tkinter is Python's de-facto standard GUI (Graphical User Interface) package -an object-oriented layer on top of Tcl/Tk.
- *To describe something as clever is NOT considered a compliment in the Python culture. Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favour of "there should be one—and preferably only one—obvious way to do it".*



Python Channel Access

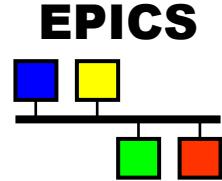
caTools



- The `cothread.catools` Python library (developed by Michael Abbott at DLS) is designed to support easy channel access from Python, and makes essential use of the features of cooperative threads.
- Cothreads (“cooperative threads”) are an approach to concurrent programming where there is only one actual thread of processing, but apparently concurrent processes (or *cothreads*) cooperatively share the processor.
- Control is passed from one cothread to another when the current cothread explicitly suspends control via a call to a cothread library routine.



caTools Summary



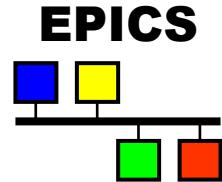
<https://cothread.readthedocs.io/en/latest/catools.html>

The caTools EPICS Channel Access Python interface consists of only three functions:

- **caget (pvs, ...)**
Retrieves value(s) from a single PV or a list of PVs.
- **caput (pvs, values, ...)**
Writes value(s) to a single PV or list of PVs.
- **camonitor (pvs, callback, ...)**
Creates a "subscription" with updates received. The callback routine is called with a new value every time any listed PV updates.



caTools : *Preliminaries*



First need to install the cothread module into a Python3 virtual environment:

```
$ dls-python3 -m venv exercise_venv  
$ source exercise_venv/bin/activate  
(exercise_venv)$ pip install cothread
```

Run a Python script in the virtual environment:

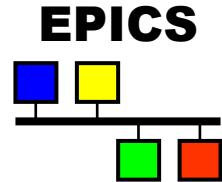
```
(exercise_venv)$ python <script_name>.py
```

Deactivate the virtual environment

```
(exercise_venv)$ deactivate  
$
```



caTools: *Preliminaries*



Now need to import cothread into our script. All of the cothread examples will start with the following code:

```
import cothread  
from cothread.catools import *
```

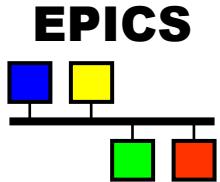
Channel access waveform data is returned as numpy arrays, so it will often be convenient to include this in our list of imports:

```
import numpy
```



Simple Example 1

File: caToolsExample1.py



```
import cothread
from cothread.catools import *

# Prompt the user for a new value
val=input('Enter new offset value: ')

# Get the current value
initial_value = caget('rjw:offset')

# Print out the previous value of a PV
print("Initial value = ", initial_value)

# Using caput: write 2.6 into PV. Raises exception on failure
caput('rjw:offset', val)

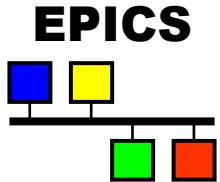
# Get the new value
new_value = caget('rjw:offset')

# Print out the new value
print("New value = ", new_value)
```



Simple Example 2

File: *caToolsExample2.py*



```
import cothread
from cothread.catools import *

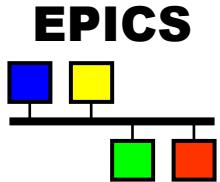
# Construct a list of PV names
bpms = ['SR%02dC-DI-EBPM-%02d:SA:X' % (c+1, n+1)
        for c in range(24) for n in range(7)]

# Get the values of the list of PVs
sax = caget(bpms)
print(sax)
```



Simple Example 3

File: caToolsExample3.py



```
import cothread
from cothread.catools import *

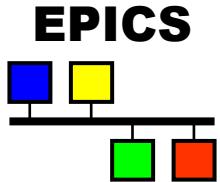
# Monitor PV, printing out each update as it is received
def print_update(value):
    print(value.name, value)
camonitor ('rjw:function', print_update)

# Now run the camonitor process until interupted by Ctrl-C
cothread.WaitForQuit()
```



Simple Example 4

File: *caToolsExample4.py*



```
import cothread
from cothread.catools import *

# Prompt the user for a new value
pos=input('Enter new motor position: ')

# Monitor readback position PV, printing out each update as it is
# received
def print_update(value):
    print(value.name, value)
camonitor ('rjw:simMotor.RBV', print_update)

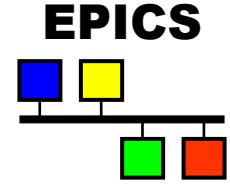
# caput the new motor position. Do callback when the motion is complete
def motor_ca_callback(value):
    print(value.name, 'Motor put callback complete')
    caput('rjw:simMotor.VAL', pos, timeout=60, callback=motor_ca_callback)

# Now run the camonitor process until interupted by Ctrl-C
cothread.WaitForQuit()
```



Python Channel Access

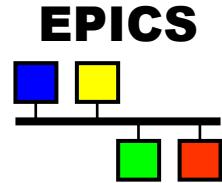
aioca



- The aioca Python library (developed by Tom Cobb at DLS) is an asynchronous EPICS Channel Access client for asyncio and Python. It is designed to support easy channel access from Python, and makes essential use of the features of asyncio to write concurrent code using the `async/await` syntax.
- Asyncio (“Asynchronous Input Output”) uses a similar approach as cothread – “cooperative multitasking”.
- Utilises an event loop to handle the scheduling of asynchronous tasks instead of using threads or subprocesses.



aioca Summary



<http://dls-controls.github.io.aioca/master/index.html>

The aioca EPICS Channel Access Python interface consists of four functions:

- **caget (pvs, ...)**

Retrieves value(s) from a single PV or a list of PVs.

- **caput (pvs, values, ...)**

Writes value(s) to a single PV or list of PVs.

- **camonitor (pvs, callback, ...)**

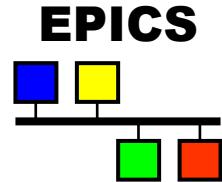
creates a "subscription" with updates received. The callback routine is called with a new value every time any listed PV updates.

- **connect(pvs, ...)**

optionally can be used to establish PV connection before using the PV.



aioca: *Preliminaries*



Again, need to install the aioca module into our Python3 virtual environment:

```
(exercise_venv)$ pip install aioca
```

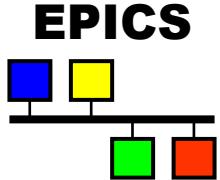
Now can import the tools we need from aioca into our script:

```
from aioca import caget, caput, camonitor, run
```



Simple Example 5

File: aiocaExample1.py



```
# Import module
from aioca import caget, caput, run

# Prompt the user for a new value
val=input('Enter new offset value: ')

async def do_stuff():
    # Print out the value of a PV
    print("Initial value = ",await caget('rjw:offset'))

    # Using caput: write value into PV. Raises exception on
    # failure
    await caput('rjw:offset', val)

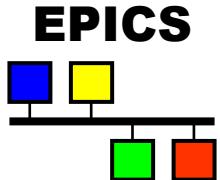
    # Print out the value of the PV we just changed
    print("New value = ", await caget('rjw:offset'))

# Now run
run(do_stuff())
```



Simple Example 6

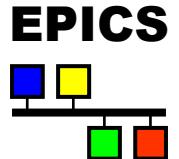
File: aiocaExample2.py



```
# Import module
from aioca import camonitor, run

pv_name = 'rjw:function'
async def do_stuff():
    # Monitor PV, printing out each update as it is received.
    def callback(value):
        print(pv_name, value)
    camonitor(pv_name, callback)

# Now run the camonitor process until interrupted by Ctrl-C.
run(do_stuff(), forever=True)
```

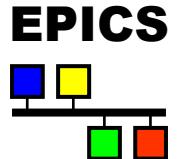


Device Support and ASYN

Andy Foster
Observatory Sciences Limited



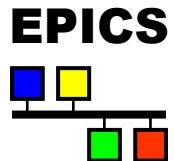
Outline



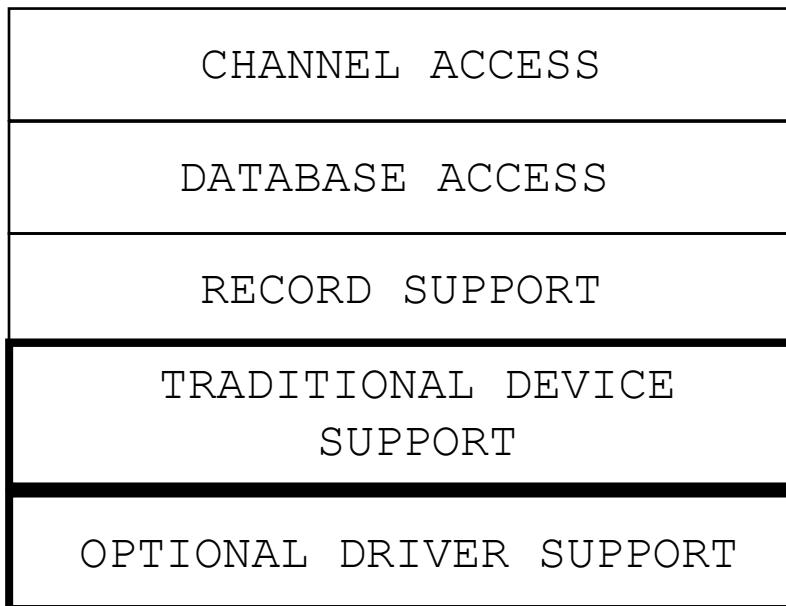
- What is Device Support?
 - Device Support Entry Table
 - dbd file entry for Device Support
 - How does a record find it's device support?
- ASYN Software: Motivation
- What is ASYN?
- ASYN Architecture
- Vocabulary
 - Asyn Port
 - Asyn Interfaces
 - asynUser
 - Asyn Command
 - AsynManager
- Synchronous control flow
- Asynchronous control flow
- Writing ASYN device support
- Asyn Trace
- asynRecord
- Where to find more information



What is Device Support?

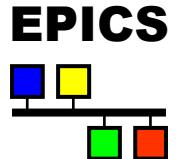


- Interface between record and hardware
- Provides an API for record support to call
- Each record type (ai, bo etc) has several device support entry tables it can call into.





The Device Support Entry Table



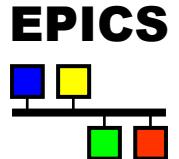
- What is a “Device Support Entry Table”?
- It’s a set of function pointers:

```
struct xxxdset {  
    long      number;  
    DEVSUPFUN report;  
    DEVSUPFUN init;  
    DEVSUPFUN initRecord;  
    DEVSUPFUN get_ioint_info;  
    DEVSUPFUN read_ai; /*record specific*/  
};
```

- *number* = number of function pointers (5)
- If a function is not needed, we can assign NULL
- Writing device support, means writing these functions to enable communication with the hardware, either with or without a driver layer.



Device Support and the .dbd file entry



- The IOC discovers what device supports are present from entries in the .dbd file

```
device(recType, addrType, dset, "name")
```

- *addrType* is one of

AB_IO	BITBUS_IO	BBGPIB_IO
CAMAC_IO	GPIB_IO	INST_IO
RF_IO	VME_IO	VXI_IO

- *dset* is the 'C' symbol name for the Device Support Entry Table (DSET)

- By convention the *dset* name indicates the record type and hardware interface:

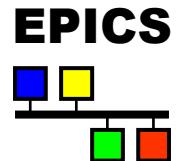
(DTYP field)

```
device(bi, INST_IO, devBiXy2440, "XYCOM-2440")
```

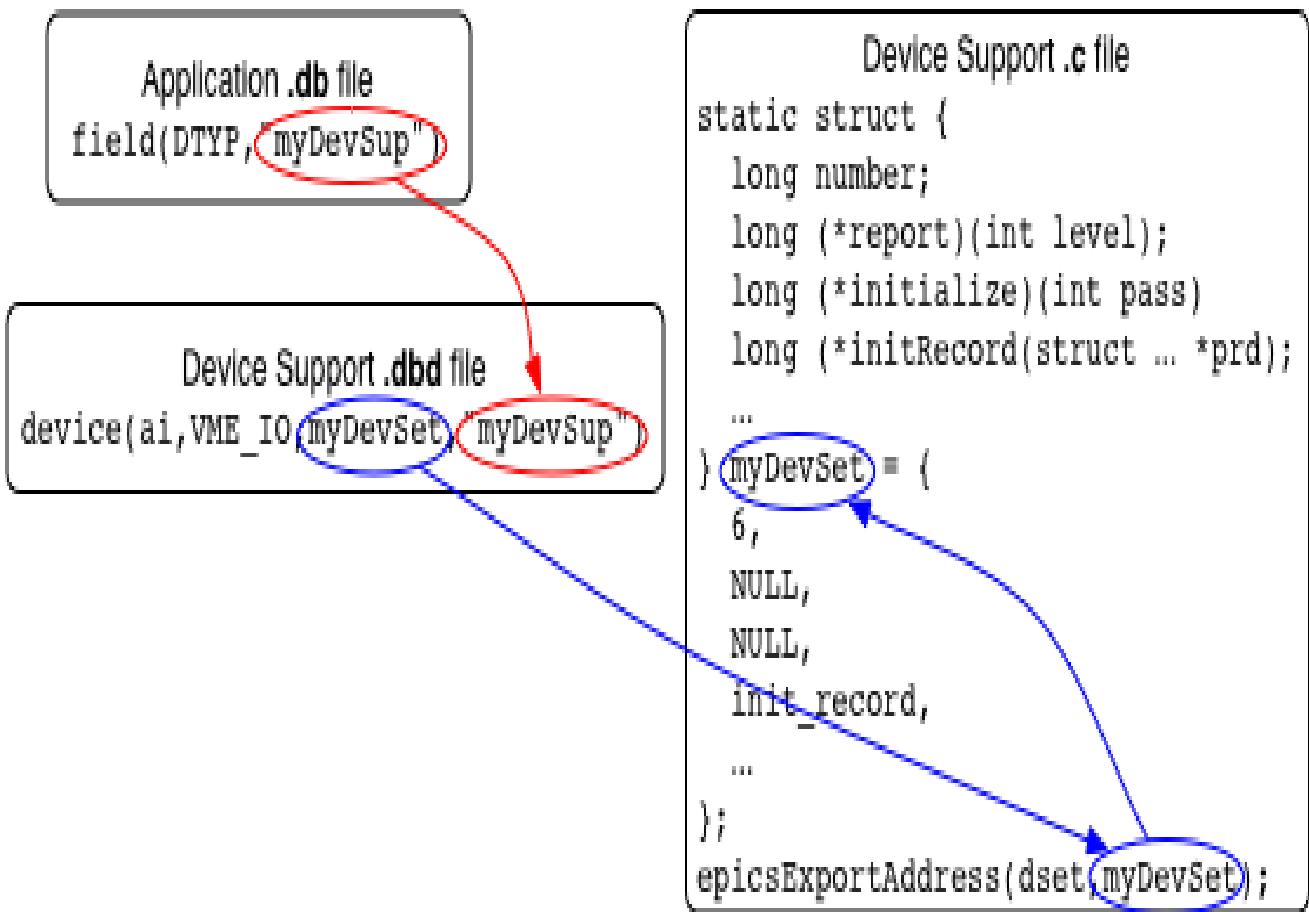
```
device(bo, VME_IO, devBoXy240, "XYCOM-240")
```



How does a record find its device support?

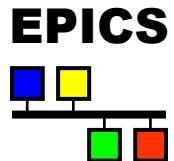


Through .dbd ‘device’ statements:

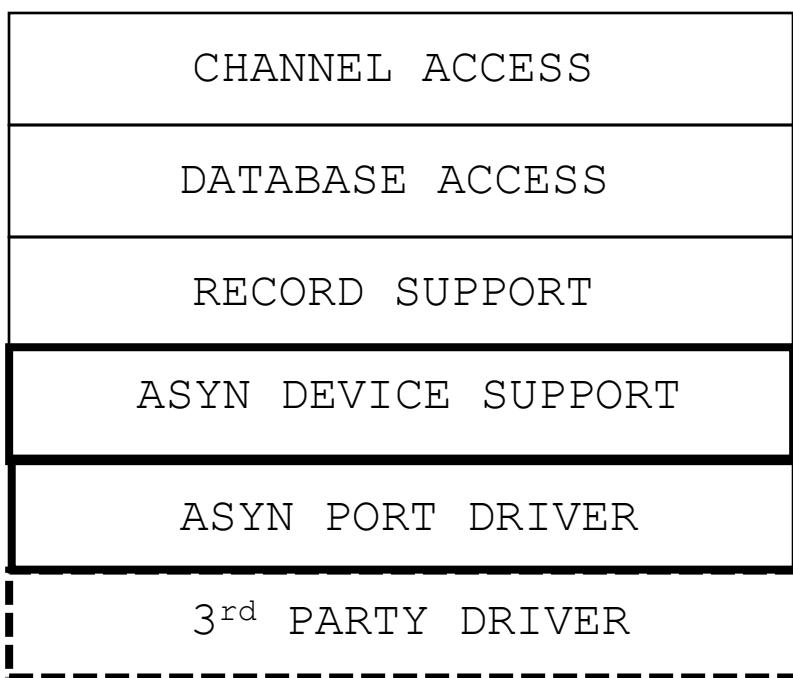




ASYN Software Motivation



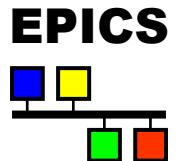
- Before the ASYN software
 - Interface between device and driver support was only loosely defined



- Historical name
 - supports “synchronous” as well as “asynchronous” devices



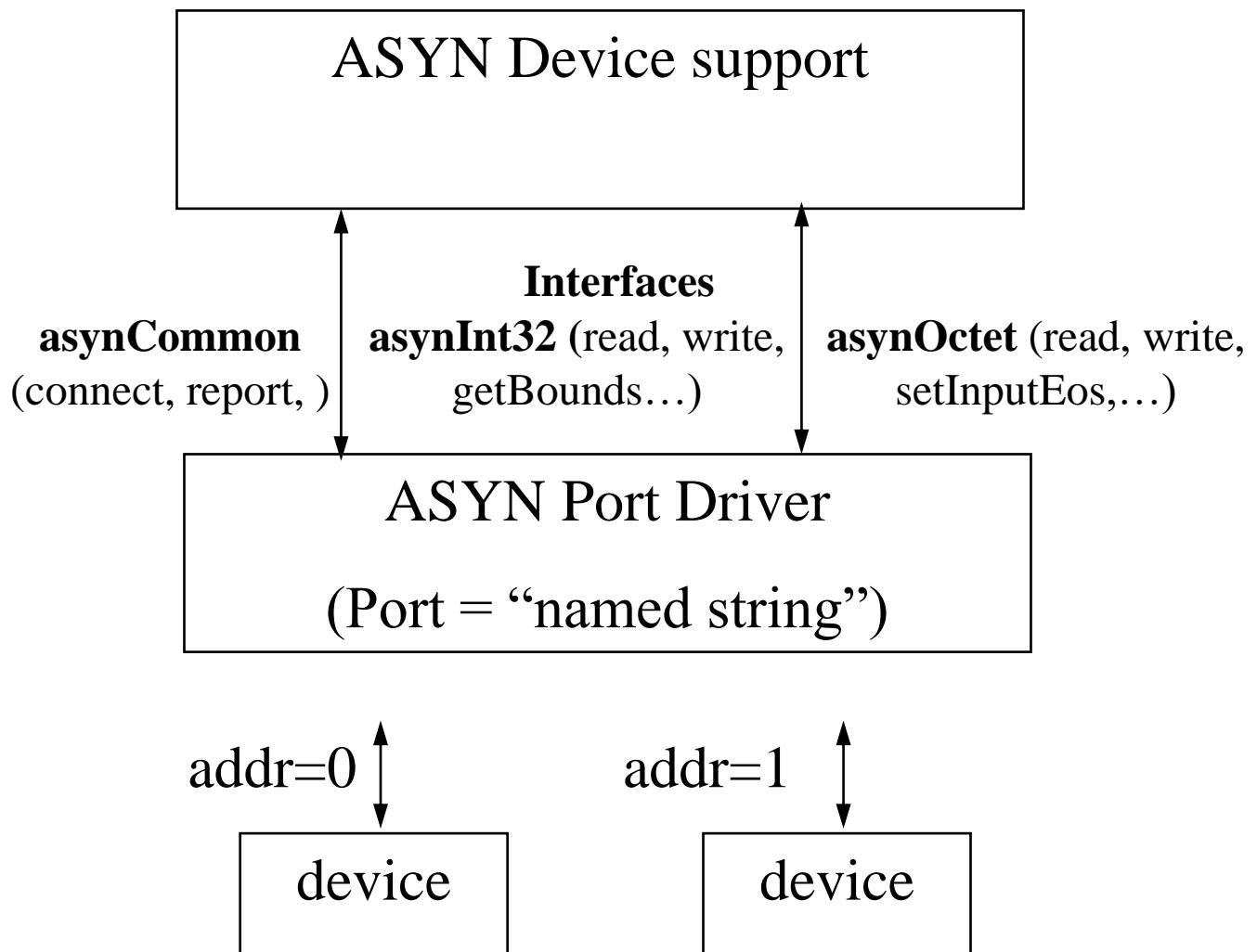
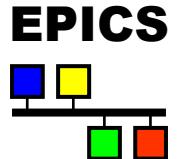
What is ASYN?



- ASYN is a software module which provides facilities for interfacing device and driver support
- Two parts:
 - **ASYN Device Support**
 - The DSET has been written for all standard EPICS records: bi, bo, mbbi, mbbo, ai, ao, waveform
 - Never need to write ASYN Device Support unless you have a custom record
- **ASYN Port Driver**
- A set of standard interfaces have been defined for:
 - message passing using ASCII strings (e.g. serial)
 - register reading/writing (e.g. DAC based devices)

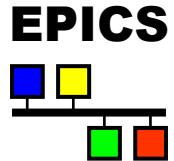


ASYN Architecture





Vocabulary: ASYN Port



- Provides access to a device
- portName (string) provides the reference to the hardware
- ASYN Drivers register a port, ASYN device support connects to the port
- One or many devices can be connected, addresses identify individual devices
- May be blocking (asynchronous) or non-blocking (synchronous)
 - Depends on speed of device
- Configured in IOC startup script:

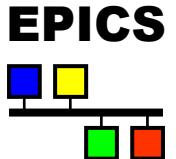
```
drvAsynSerialPortConfigure  
  ("COM2", "/dev/sttyS1")
```

```
drvAsynIPPPortConfigure  
  ("fooServer", "192.168.0.10:40000")
```

```
myDeviceDriverConfigure("portname", "params")
```



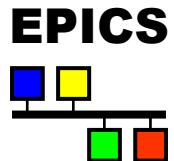
Vocabulary: ASYN Interfaces



- Device support communicate with driver support through defined **interfaces**
- Each interface defines a table of driver functions or methods
- Examples are:
 - asynOctet (read, write, setInputEOS, ...)
 - Used for message based I/O: serial, TCP/IP
 - asynInt32 (read, write, getBounds...)
 - Integer registers: ADC, DAC, encoder
 - asynInt32Array (read, write, getBounds, ...)
 - Integer arrays: spectrum analyzer, oscilloscope
 - asynFloat64, asynFloat64Array (read, write,...)
 - Floating point registers and arrays



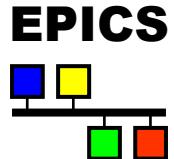
Vocabulary: ASYN Interfaces



- asynCommon (report, connect, disconnect)
 - report:
Generates a report about the hardware device
 - connect:
Connect to the hardware device
 - disconnect:
Disconnect from the hardware device
- Every driver must implement these!
- asynDrvUser(create, getType, destroy)
 - This interface is used to support commands sent to the driver from device support.
- Every ASYN Port Driver has several interfaces



Vocabulary: *asynUser*



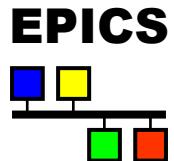
- ASYN Device Support for EPICS records creates a unique *asynUser* for each record instance
- An *asynUser* is a structure, see:
asyn/asynDriver/asynDriver.h
- We use the “asynUser” to:
 - Access ASYN ports
 - Call interfaces implemented by port drivers
- It cannot be shared: uniqueness prevents simultaneous access to the Port Driver

```
pasynUser=pasynManager->createAsynUser(  
    processCallback, timeoutCallback);
```

- Provide 2 callbacks:
- *processCallback* is called when you are scheduled to access the port. Inside this callback is where we call write and read methods
- *timeoutCallback* is called if port times out



Vocabulary: ASYN Command



- An ASYN driver defines a set of commands it supports
 - e.g. **VER** – Give me the firmware version?
- In an EPICS record the ASYN command is the **string** at the end of the INP or OUT link field.

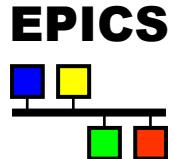
```
record(waveform, "$(device):firmware")  
{  
    field(DTYP, "asynOctet")  
    field(INP, "@asyn(port, addr, timeout) VER")  
}
```

port = ASYN port string,
addr = which device on port?

- The DTYP field is set to the type of ASYN interface being used



Vocabulary: *asynManager*



- Core of ASYN.
- Creates threads for blocking ports.
- Methods to:
 - connect to a driver port
 - find driver interfaces.
- Schedules access to ports.
 - Device support and driver support do not need to implement queues or semaphores, this is handled by *asynManager*.
- Clients ask *asynManager* for services

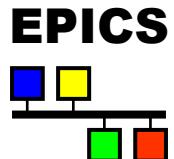
```
pasynManager->connectDevice(pasynUser ,  
"portname", address)
```

```
pasynManager->findInterface(pasynUser,  
interfaceType, ...)
```

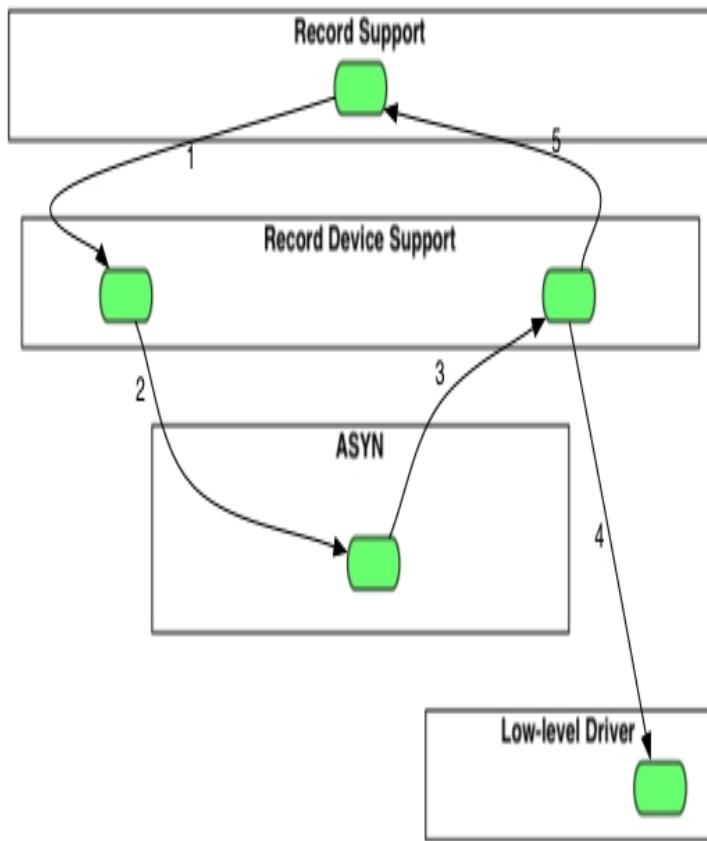
```
pasynManager->queueRequest(pasynUser,  
priority, timeout)
```



Control flow for non-blocking port



All code runs in application thread

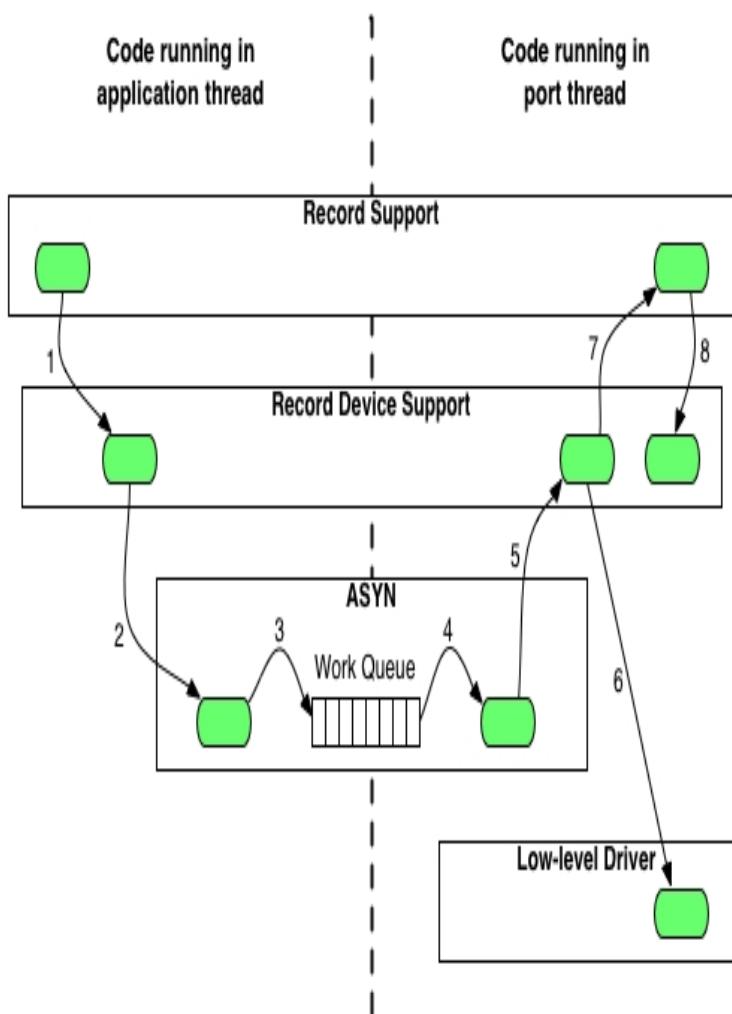
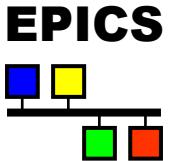


1. Record processing calls device support.
2. Device support calls queueRequest.
3. Since the port is synchronous, queueRequest locks the port and calls "processCallback" immediately
4. "processCallback" calls the driver *read*, *write* methods and then returns to "queueRequest".
5. "queueRequest", unlocks the port, and returns to device support and record processing.

Used for devices that provide fast responses, typically VME or register based devices



Control flow for blocking port

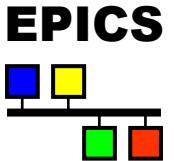


1. Record processing calls device support with PACT=0.
2. Device support sets PACT=1 calls queueRequest.
3. queueRequest places request on queue (app. thread now continues).
4. Port thread removes the request from the queue.
5. Port thread calls “processCallback”
6. “processCallback” calls driver *read*, *write* and blocks until the operation is complete. Returns to “processCallback”.
7. “processCallback” calls the EPICS routine “callbackRequestProcessCallback” to make the record process again.
8. Record support calls device support again. But now PACT=1. Device support updates fields in the record (sets PACT=0) and returns to record support.

Used for ‘slow’ devices like serial or ethernet



Writing ASYN Device Support Initialisation



Remember: One of our DSET routines was *initRecord*.
What must the code in here do?

- 1) Obtain a **pointer to an asynUser**

```
pasynUser = pasynManager->createAsynUser(  
    processCallback, timeoutCallback);
```

- 2) Connect to the device (port, address)

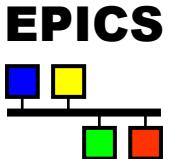
```
status=pasynManager->connectDevice(pasynUser,  
    port, addr);
```

- 3) Find the interface (e.g. asynOctet)

```
pasynInterface=pasynManager->findInterface  
(pasynUser, asynOctetType, 1);
```



Writing ASYN Device Support Initialisation



4) From the *pasynInterface* structure:

 find the address of the asynOctet interface

```
pasynOctet=(asynOctet *)pasynInterface->pinterface;
```

5) Access the command string, at the end of the INP link.

Record has: *field(INP, "@asyn(port, addr, timeout) VER")*

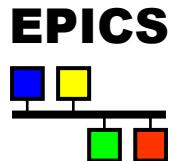
6) Call the *asynDrvUser* interface *create* method to:

- (a) map the “reason” member of the *asynUser* structure to the command this record is sending to the driver
- (b) Needed for the record processing routine

```
status = pasynDrvUser->create(void *pvt,  
                               pasynUser, const char *cmdstring,...);  
{  
    if (cmdstring)  
    {  
        if (strcmp("VER", cmdstring) == 0)  
            pasynUser->reason = 1;  
        else if(strcmp("XXX", cmdstring) == 0)  
            pasynUser->reason = 2;  
    }  
}
```



Writing ASYN Device Support Process Routine



- Ask asynManager to put your request on the queue:

```
status=pasynManager->queueRequest(pasynUser,  
                                     priority, timeout);
```

Remember: asynUser has a processCallback routine

Priorities from record configuration (PRIO)
asynQueuePriority{Low | Medium | High}

queueRequest call never blocks.

(a) Blocking port:

 asynManager calls processCallback when port is free.
 Callback runs in port thread.

(b) Non blocking port:

 queueRequest calls processCallback immediately

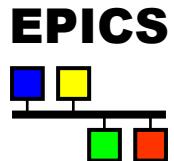
If port busy for timeout seconds, asynManager calls timeoutCallback.

In processCallback, you have exclusive access to the port.



Writing ASYN Device Support

processCallback (*asynOctet* methods)



- When your *processCallback* is called, use methods from the *asynOctet* interface:

- Flush (discard old input)

```
status=pasynOctet->flush( drvPvt, pasynUser );
```

- Write:

```
status = pasynOctet->write(  
    drvPvt, pasynUser, data, size, &bytesWritten);
```

Number of bytes written is returned in *bytesWritten*.

- Read:

```
status=pasynOctet->read(drvPvt, pasynUser, buffer,  
    maxsize, &bytesReceived, &eomReason);
```

Number of bytes read is returned in *bytesReceived*.

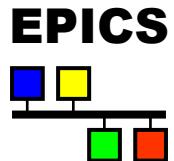
End of message reason is returned in *eomReason*.



Writing ASYN Device Support

processCallback

(asynInt32 methods)



- Or, if you searched for the asynInt32 interface, use methods from it:

□ Write

```
status=pasynInt32->write(  
                           drvPvt, pasynUser, value);
```

□ Read

```
status=pasynInt32->read(  
                           drvPvt, pasynUser, &value);
```

- Current register value is returned in value.

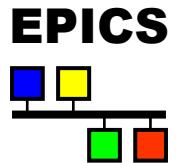
□ Get bounds

```
status=pasynInt32->getBounds(  
                           drvPvt, pasynUser, &low, &high);
```

- Limits for valid register values are returned in low and high.



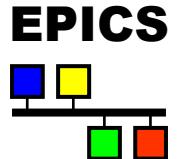
Rules for using driver methods



- Never use I/O methods outside `processCallback`.
- Only talk to the port that has called you back.
- You can do as much I/O as you like.
- You must always use the interface method table `pasyn{Octet | Int32 | ...}` to access the driver.
- You always need `pasynUser` as an argument.
- All other clients of the same port (even with other addresses) have to wait until you are finished. So, remember, it's not nice of you if your device blocks for a long time!



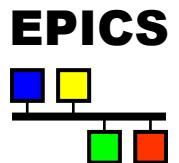
Vocabulary: *asynTrace*



- Diagnostic facility
 - Provides routines to call for diagnostic messages: *asynPrint()*, *asynPrintIO()*
 - Several masks or levels can be selected for debugging purposes
 - Provides consistent debugging mechanism for drivers



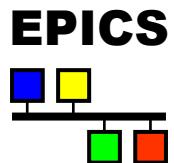
asynRecord



- Special record type that can use all asyn interfaces.
- Can dynamically connect to different ports at run-time.
- Is a good debug tool.
- Access to options, including tracing.
- Comes with set of edm, medm, CSS screens
- Can handle simple devices:
 - e.g. asynOctet: write one string, read one string
- If a new instrument arrives that has a serial or ethernet port, communicate with:
 - A database containing a single asynRecord!



asynRecord medm screens



X asynRecord.adl

13LAB:serial17

Port: serial17 Address: 0

Connect Connected

drvInfo: Reason: 0

Interface: asynOctet

Cancel queueRequest More...

Error:

Connected Enabled autoConnect

Connect Enable autoConnect

traceMask	traceIOMask
0x1	0x0
Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceError	Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceIOASCII
Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceIODevice	Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceIOEscape
Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceIOFilter	Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceIOHex
Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceIDriver	80 Truncate size
Off <input type="checkbox"/> On <input checked="" type="checkbox"/> traceFlow	
Trace file: Unknown	

X asynOctet.adl

13LAB:serial17

Timeout (sec): 1.000 Transfer: Write/Read

asynOctet interface: Supported Active

Output Format: ASCII Terminator: \r

ASCII: tptptp

Length: Requested: 80 Actual: 6

Input Format: ASCII Terminator: \r

ASCII: ITP30.001,2TP0.000,3TP=0.001,4TP0.000

Length: Requested: 0 Actual: 37

EOM reason: Eos

I/O Status: NO_ALARM I/O Severity: NO_ALARM

Scan: Passive Process More...

X asynRegister.adl

13LAB:serial17

Timeout (sec): 1.000 Transfer: Read

Interface: Int32 UInt32Digital Float64

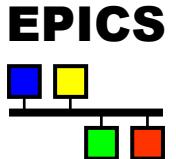
asynInt32 <input type="checkbox"/>	Supported <input type="checkbox"/>	Unsupported <input type="checkbox"/>	Supported <input type="checkbox"/>
	Active <input type="checkbox"/>	Inactive <input type="checkbox"/>	Inactive <input type="checkbox"/>
Output: 0 <input type="button"/>	0 <input type="button"/>	0 <input type="button"/>	
Output (hex): 0x0 <input type="button"/>	0x0 <input type="button"/>	0x0 <input type="button"/>	
Input: 32769 <input type="button"/>	0 <input type="button"/>	0 <input type="button"/>	
Input (hex): 0x8001 <input type="button"/>	0x0 <input type="button"/>		
Mask (hex): <input type="button"/>	0xffffffff <input type="button"/>		

I/O Status: NO_ALARM I/O Severity: NO_ALARM

Scan: .1 second Process More...



More information

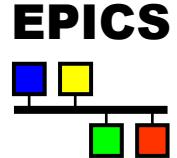


- Talk about asynDriver
 - www.aps.anl.gov/epics/docs/USPAS2007.php
- AsynDriver
 - www.aps.anl.gov/epics/modules/soft/asyn/
- Drivers/device supports using asynDriver
 - www.aps.anl.gov/aod/bcda/synApps/
- StreamDevice
 - epics.web.psi.ch/software/streamdevice/

EPICS **StreamDevice Support**

Philip Taylor
Observatory Sciences Ltd

EPICS StreamDevice



- Generic EPICS device support for devices with a "byte stream" based communication interface
- Byte stream devices are those that can be controlled by sending and receiving strings
- Based on the EPICS Asyn library
- Written and maintained by Dirk Zimoch at PSI (Swiss Light Source) Switzerland

“Stream” Type Protocols

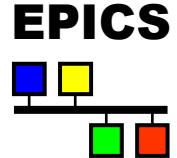
- Example Devices:
 - Temperature controllers: Lakeshore, Omega, ...
 - Vacuum pumps & Gauges: Varian, ...
- Serial (RS-232, RS485), Networked (TCP), GPIB (IEEE-488)
- Text-based command / response communication
- Example (Lakeshore Temperature Controller):

What is the current temperature on channel A?

Command: “**KRDG? A\n**”

Response: “**+077.350E+0\n**” (*It’s about 80 Kelvin*)

Simple Example: Protocol File



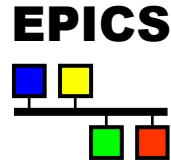
File *demo.proto*

```
Terminator = CR;           Input and output are both terminated by CR (\r)

getTempA                  Name of a protocol
{
    out "KRDG? A";        Protocol body in braces {}
    in "%f";              Output written to device
}
                                Read and parse input from device
```

- Protocol file is called *xxx.proto*, where *xxx* is the name of the device
- At Diamond, this file is kept in: “*exerciseApp/protocol*”

Simple Example: Database



ai Record

```
record(ai, "Temp:B")
{
    field(DTYP,"stream")
    field(INP,"@demo.proto getTempA TC1")
    field(SCAN,"5 second")
}
```

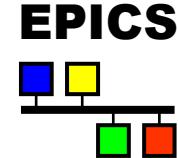
Name of protocol file

Specific protocol

Asyn Port

- The database file lives in: “*exerciseApp/Db*”

Simple Example: IOC Config



In the IOC startup script (for our exercises, this will be “`iocBoot/iocexercise/stexercise7/src`”) you will need a line like this:

```
drvAsynIPPortConfigure ("TC1", "192.168.164.10:23")
```

Asyn port referenced in the record's INP/OUT field



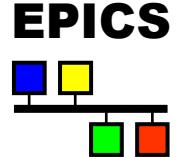
IP address of device : Port number

System Variables

System Variables are used to set defaults for all protocols, they include:

Variable	Meaning
Terminator InTerminator OutTerminator	Can separately specify input and output terminators
ReadTimeout WriteTimeout	Timeouts in millisecs
MaxInput	Terminate input on fixed message size
Separator	Define separator for arrays of values
ExtraInput	Error or Ignore. Define how unexpected input is handled

Protocol Arguments



- It is often convenient to write only one protocol and use *protocol arguments* for the differences
- For example, a motor controller for the 3 axes X, Y, Z requires three protocols to set a position.

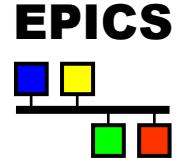
```
moveX { out "X GOTO %d"; }  
moveY { out "Y GOTO %d"; }  
moveZ { out "Z GOTO %d"; }
```

- A single move protocol could be written with a protocol argument:

```
move { out "\$1 GOTO %d"; }
```

- The OUT links for 3 records could then be written as move(X), move(Y) and move(Z)
- Up to 9 parameters \$1...\$9 are supported

Format Convertors



- StreamDevice *format converters* are very similar to the format converters used in the C functions printf() and scanf(), but also provide some additional options and convertors
- Examples:

in "%f";	<i>float</i>
out "%7.4f";	<i>7 char float with precision 4</i>
out "%#010x";	<i>0-padded 10 char hex (with leading 0x)</i>
in "%[_a-zA-Z0-9]";	<i>string of chars out of a charset</i>
in "%*i";	<i>skip input integer number</i>
in "%?d";	<i>decimal number or nothing (default 0) if failed</i>
in "%=.3f";	<i>compare input to the current value formatted as a float with precision 3</i>
out "%<crc8>"	<i>8-bit CRC checksum. About 15 other checksum types are supported.</i>

Redirection

- By default, the record's VAL field is used for data input/output, but *redirection* allows I/O to other fields of the record or even other records
- Simple examples:

out "%(EGU)s"

Output EGU field formatted as a string

in "%(record.RVAL)f"

Read floating point value into the RVAL field of another record

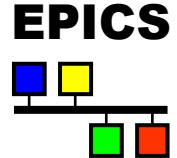
- Get multiple values into multiple records, using protocol parameter:

```
read_AB {out "GET A,B"; in "A=%f, B=%(\${1})f"; }

record (ai, "DEVICE:A") {
    field (DTYP, "stream")
    field (INP, "@mydevice.proto read_AB(DEVICE:B) ${BUS}")
    field (SCAN, "1 second")
}

record (ai, "DEVICE:B") {
    field (SCAN, "Passive")
}
```

Supported EPICS Record Types



- StreamDevice comes with comprehensive support for all standard record types in EPICS base which provide device support
- Online documentation provides details about the supported types
 - aai aao ai ao bi bo mbbi mbbo mbbiDirect
 - mbboDirect longin longout stringin stringout
 - waveform calcout scalcout
- Each page describes details of the handling of each record type: which record fields are used in input and output for different format data types during normal record processing and initialization.
- Note: I/O event scanning is supported with the SCAN field set to “**I/O Intr**”. This is useful for handling devices that send data automatically.

Another Protocol File Example

Init handlers
set the initial
value for these
records

```

Terminator = CR LF;

# Frequency is a float, so use ai and ao records
getFrequency {
    out "FREQ?"; in "%f";
}
setFrequency {
    out "FREQ %f";
    @init { getFrequency; }
}

# Switch is an enum (either OFF or ON) so use bi and bo records
getSwitch {
    out "SW?"; in "SW %{OFF|ON}";
}
setSwitch {
    out "SW %{OFF|ON}";
    @init { getSwitch; }
}

# Connect a stringout record to this to get a generic command interface.
# After processing finishes, the record contains the reply.
debug {
    ExtraInput = Ignore;
    out "%s"; in "%39c"
}

```

Adding support to IOC

- `configure/RELEASE`:

```
SUPPORT= /dls_sw/prod/R3.14.12.7/support  
ASYN=      $(SUPPORT)/asyn/4-41  
STREAM=    $(SUPPORT)/streamDevice/2-5dls13
```

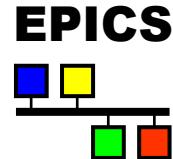
- `demoApp/src/Makefile`:

```
demo_DBD += stream.dbd  
demo_DBD += asyn.dbd  
demo_DBD += drvAsynIPPort.dbd
```

```
demo_LIBS += asyn  
demo_LIBS += stream
```

- If you look in “`exerciseApp/src/Makefile`” you will see these lines but with “`demo`” replaced by “`exercise`”.

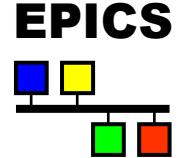
IOC Startup File



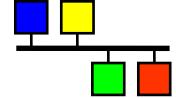
```
epicsEnvSet ("STREAM_PROTOCOL_PATH", "$(DEMO)/data")  
  
# Following line used for TCP/IP port connection  
  
drvAsynIPPortConfigure ("NC", "127.0.0.1:6543")  
  
# Following lines for a serial connection  
  
drvAsynSerialPortConfigure("LPORT0", "/dev/ttyS0")  
asynSetOption("LPORT0", 0, "baud", "9600")  
asynSetOption("LPORT0", 0, "bits", "7")  
asynSetOption("LPORT0", 0, "parity", "odd")  
asynSetOption("LPORT0", 0, "stop", "1")  
  
# Log some asyn info and in/out texts  
  
asynSetTraceMask("NC", 0, 4)  
asynSetTraceIOMask("NC", 0, 6)  
  
dbLoadRecords("db/stream.db","user=fred")
```

# ASYN_TRACE_ERROR	0x0001
# ASYN_TRACEIO_DEVICE	0x0002
# ASYN_TRACEIO_FILTER	0x0004
# ASYN_TRACEIO_DRIVER	0x0008
# ASYN_TRACE_FLOW	0x0010
# ASYN_TRACEIO_NODATA	0x0000
# ASYN_TRACEIO_ASCII	0x0001
# ASYN_TRACEIO_ESCAPE	0x0002
# ASYN_TRACEIO_HEX	0x0004

What StreamDevice does for you



- Handles the connection to the device
 - Can re-connect automatically after disconnection
- Allows many records to communicate via one connection
 - Threading, queuing, ...
- Handles timeouts, errors
 - Put records into ‘alarm’ state
- Provides debug options
 - Log every byte sent/received



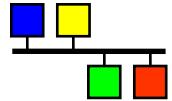
CSS/BOY & Archive Viewer

Philip Taylor & Andy Foster
(Observatory Sciences Ltd)

February 2021



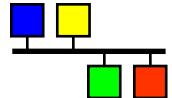
Control System Studio (CSS-RCP)



- Control software user interface framework with an emphasis on interoperability
- Being developed at DESY/ORNL/BNL
- Java/Eclipse based
- Not EPICS specific
 - Data Access Layer (DAL) provides transparent access to specific communication protocols
 - EPICS , TANGO and TINE (DESY) layers have been implemented
- Aim is to produce an integrated toolset providing a common view of data (e.g. EPICS PVs) and a common look and feel



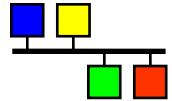
Control System Studio (CSS)



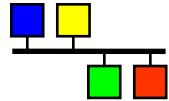
- Source code available via GitHub at:
<https://github.com/ControlSystemStudio>
- Pre-built distributions available from DESY, SNS and others, see SNS website at:
<http://ics-web.sns.ornl.org/css>
- CSS Tools produced so far include
 - PV Tree – displays link hierarchy of EPICS records
 - ELOG – Electronic Logbook
 - Data Browser
 - BOY (Best Operator interface Yet)
 - BEAUTY (Best Ever Archive Toolset)
 - BEAST (Best Ever Alarm System Toolkit)



Control System Studio (CSS)



- Uses the Eclipse IDE with RCP (Rich Client Platform)
- Makes use of an extensive range of standard Java facilities and components
- Provides RDB access via the JDBC standard interface (MySQL, PostgreSQL or Oracle)
- Uses standard Java Messaging Service (JMS) to send messages (e.g. log) between CSS applications.
- The following screens are the CSS “Example Use case” provided by ORNL



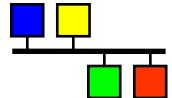
CSS Use Case/1

Alarm Table							
Current Alarms		Filter					
PV	Description	Time	Current severity	Severity	Status	Value	
CF_KL:DIWS_AIT4303B:Rs	CF_KL:DIWS_AIT4303B:Rs	2009/03/17 16:10:06	MINOR	MINOR	HIGH_ALARM	18.5	
RFQ_Vac:Pump2:Pressure	Demo pump 2	2009/03/17 16:09:46	OK	MAJOR	HIHI_ALARM	9.0	
RFQ_Vac:Pump6:Pressure	Demo pump 6	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0	
RFQ_Vac:Pump5:Pressure	Demo pump 5	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0	
RFQ_Vac:Pump4:Pressure	Demo pump 4	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0	
RFQ_Vac:Pump3:Pressure	Demo pump 3	2009/03/17 16:09:44	OK	MINOR	HIGH_ALARM	5.0	
MEBT_CHOP:PS_2:V	mebbit chopper power supply two voltage fault	2009/03/16 19:05:10	MAJOR	MAJOR	LOLO_ALARM	0.000	
Acknowledged Alarms							
PV	Description	Time	Current severity	Severity	Status	Value	
TMod:Summary_MPS:Alarm	Moderator System MPS Trip	2009/03/16 19:05:09	INVALID	invalid-ack'd	READ_ALARM	Ready	
MEBT_CHOP:PS_1:V	mebbit chopper power supply one voltage f	2009/03/16 19:05:10	MAJOR	major-ack'd	LOLO_ALARM	0.000	
HEBT_Coll:CT2:Cond	HEBT_Coll:CT2:Cond	2009/03/16 19:05:10	MAJOR	major-ack'd	LOLO_ALARM	0.017	
FE_MPS:MIROC1A:status_sum	MPS Beam permit	2009/03/17 16:05:00	MAJOR	major-ack'd	LOLO_ALARM	2	
ICS_Tim:Gate_BeamOn:Switch	Beam awf	2009/03/17 16:04:59	MINOR	minor-ack'd	STATE_ALARM	Shifted	

- Operator is using the Alarm Table application.
- Power supply fault reported on PV MEBT_CHOP:PS_2:V



CSS Use Case/2



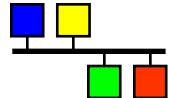
PV	Description	Time	Current Severity	Severity	Status	Value
RFQ_Vac:Pump2:Pressure	Demo pump 2	2009/03/17 16:48:10	OK	MAJOR	HIGH_ALARM	9.0
RFQ_Vac:Pump6:Pressure	Demo pump 6	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump5:Pressure	Demo pump 5	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump4:Pressure	Demo pump 4	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
RFQ_Vac:Pump3:Pressure	Demo pump 3	2009/03/17 16:48:08	OK	MINOR	HIGH_ALARM	5.0
FE_MPS:MIOC1A:status_sum	MPS Beam permit	2009/03/17 16:46:28	MINOR	MAJOR	LOLO_ALARM	2
ICS_Tim:Gate_BeamOn:Switch	Beam awf	2009/03/17 16:46:27	MINOR	MINOR	STATE_ALARM	Shift
CF_KL:DWMS_AIT4303B:Rs	CF_KL:DWMS_AIT4303B:Rs	2009/03/17 16:10:06	MINOR	MINOR	HIGH_ALARM	18.5
MEBT_CHOP:PS_2:V	mebbit chopper power supply two voltage fault		DR	MAJOR	LOLO_ALARM	0.00

PV	Description	Time
TMod:Summary_MPS:Alarm	Moderator System MPS Trip	2009/03/17 16:46:28
MEBT_CHOP:PS_1:V	mebbit chopper power supply one voltage fault	2009/03/17 16:46:27
HEBT_Coll:CT2:Cond	HEBT_Coll:CT2:Cond	2009/03/17 16:46:27

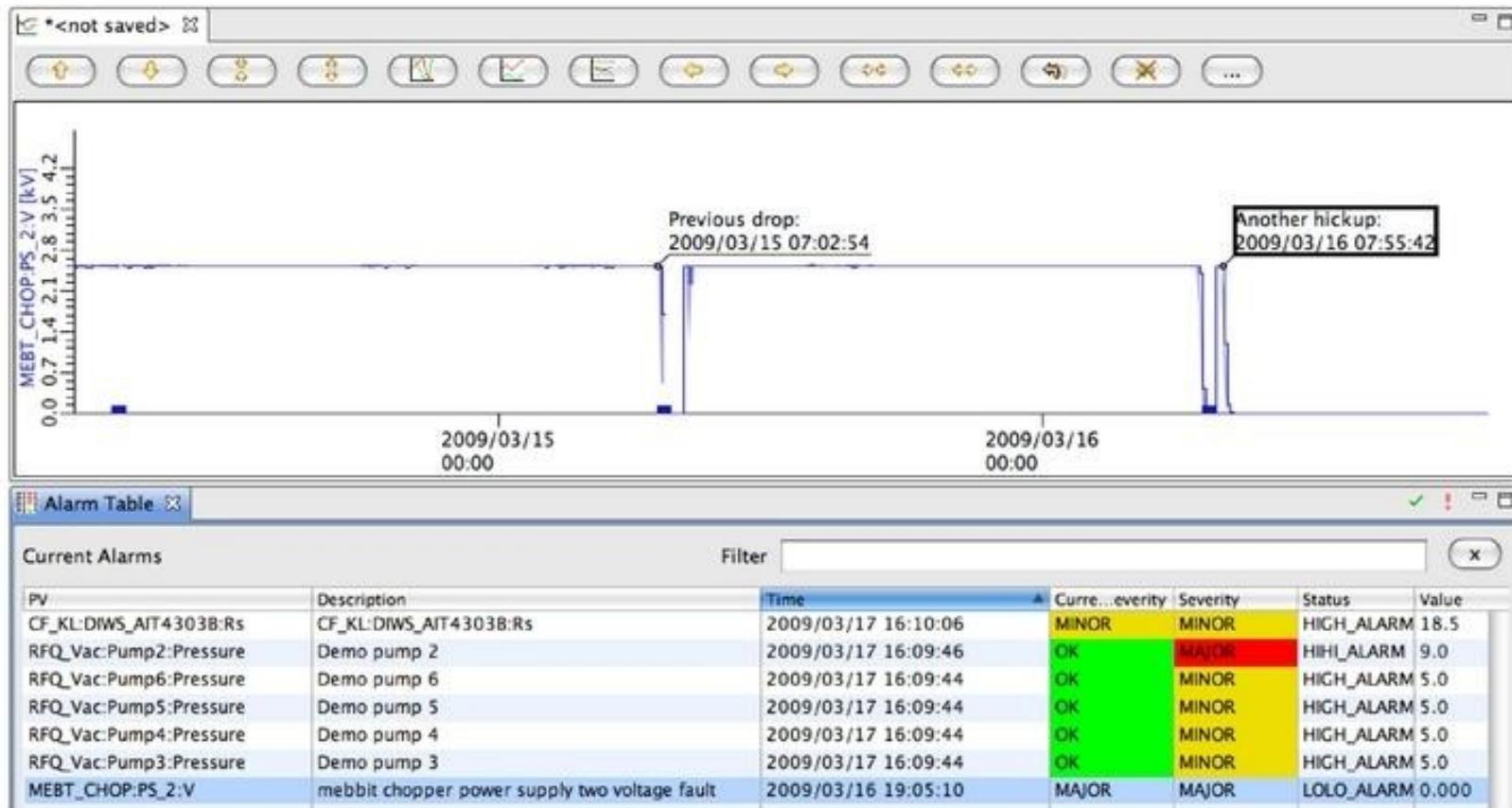
- 21:44:56
- i Check MEBT PS 2 Chopper
- MEBT Chopper PS 2 Screen
- Logbook...
- Acknowledge
- Copy Pv Name to Clipboard
- CSS**
- Configure Item
- Auto-size Columns
- Alarm Perspective

- Data Browser
- Data Browser View
- PV Table
- Rack View
- PV Utility
- PV Fields Viewer
- Probe
- EPICS PV Tree

- Operator uses the CSS menu to open the “Data Browser” for this PV



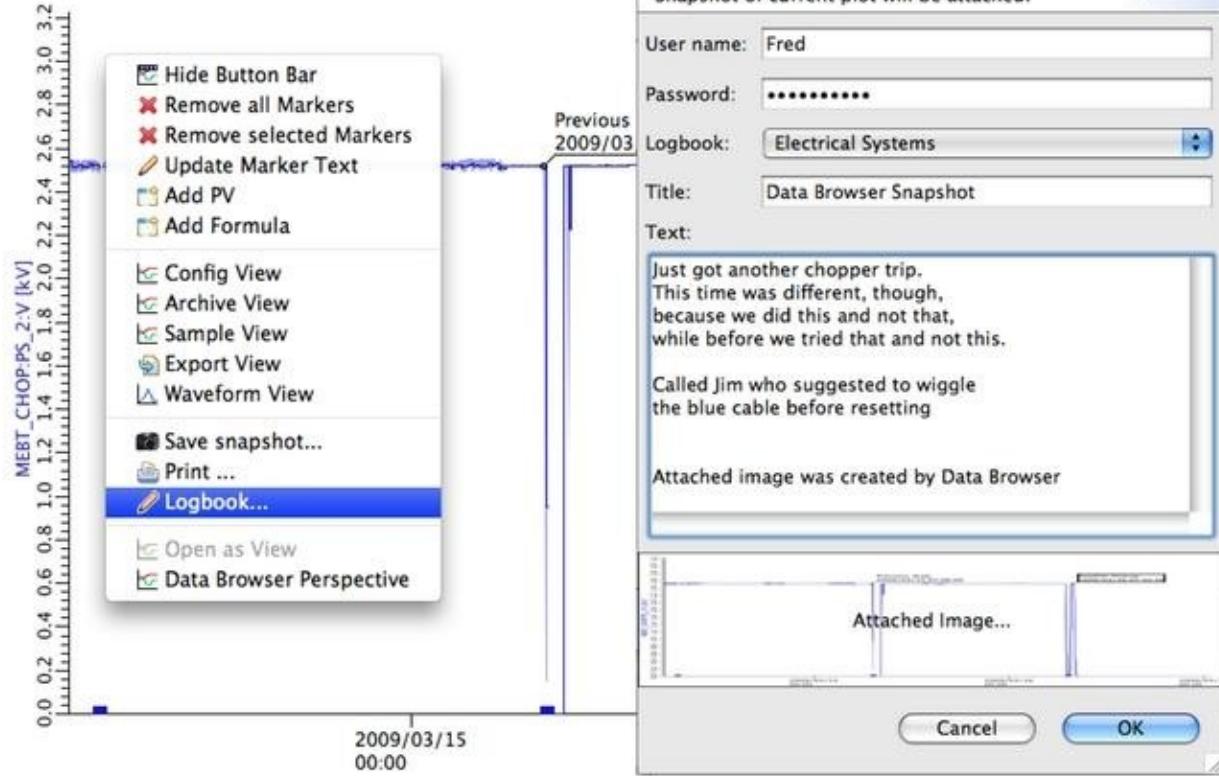
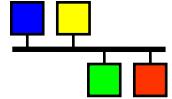
CSS Use Case/3



- Operator views and annotates the voltage drop events using the Data Browser



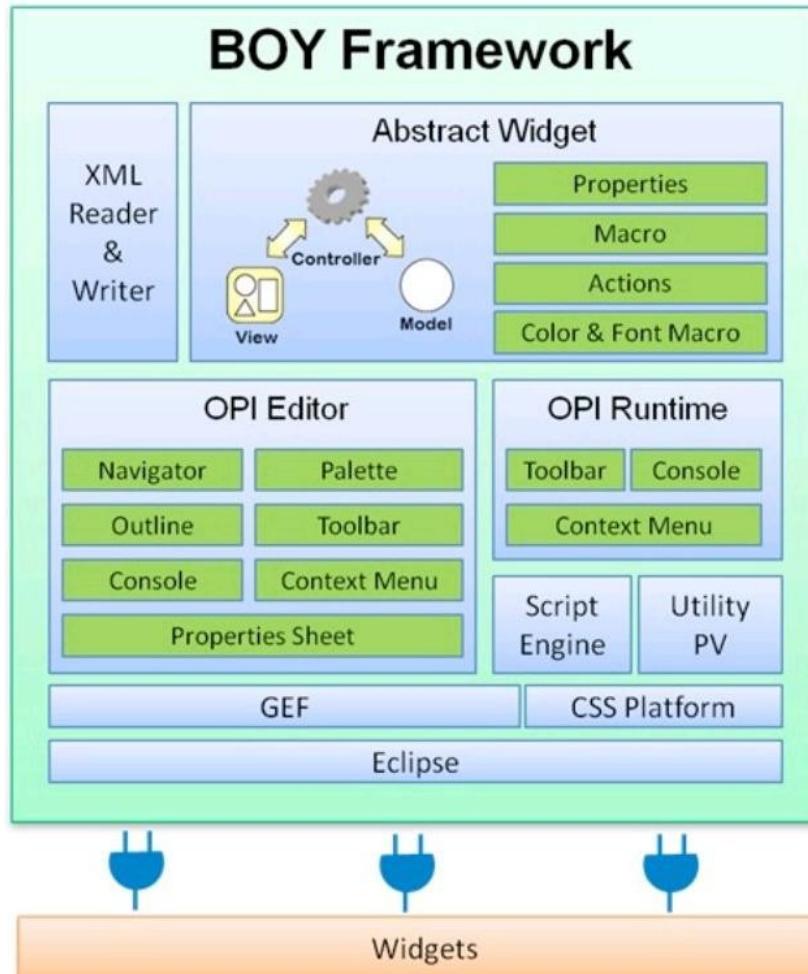
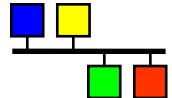
CSS Use Case/4



- Opening up the electronic Logbook, the operator enters a text log entry and attach a screenshot



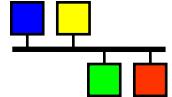
BOY - “Best OPI Yet”



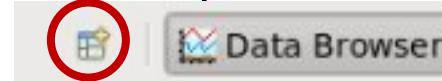
- Uses Eclipse Graphical Editing Framework (GEF)
- Widgets are included in BOY via the Eclipse plug-in mechanism
- Rules and scripts (JavaScript or Python) can be attached to a widget and can dynamically change any widget property
- Macros at many levels. Includes PV, color and font macros.
- OPI files defined using XML
- Independent data access layer



Starting BOY (@DLS)



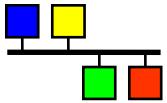
- In Diamond Launcher:
Utilities-> Control System Studio
- The appearance will depend on whether you have run it before.
- When CSS top screen opens, see top-right toolbar:



- Left click on Open Perspective, select OPI Editor
- Select from File Browser on left, CSS->(your FedID)
- Right click, select New->OPI File
- Select FedID folder
- Type a new OPI file name in the OPI File Name text box
- Click on Finish

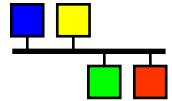


OPI Editor Overview



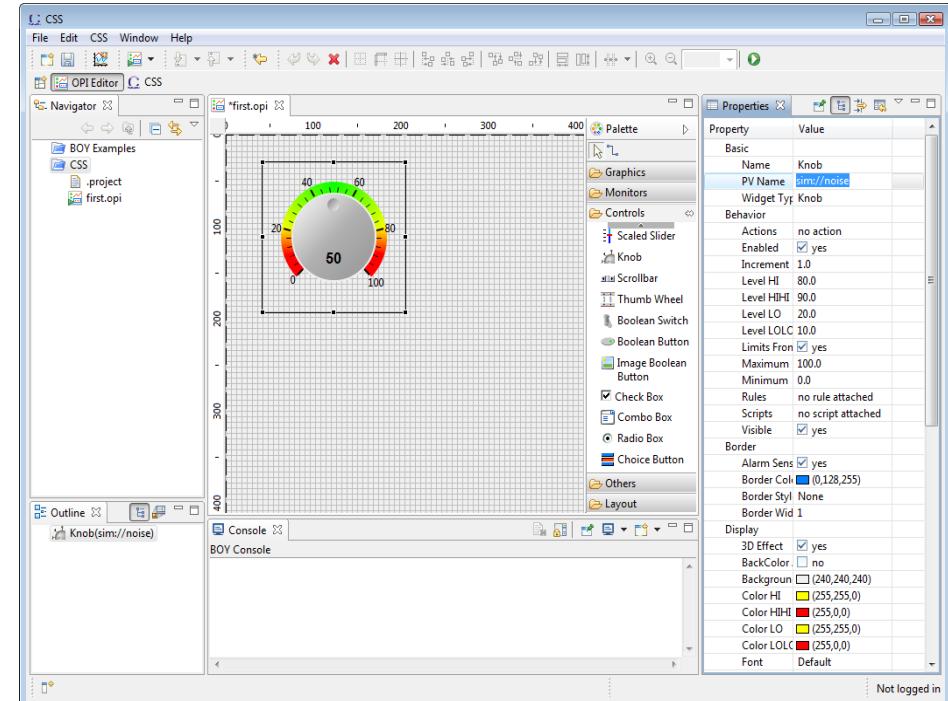
The screenshot shows the OPI Editor interface with several labeled components:

- Toolbar**: Located at the top, containing various icons for file operations.
- Navigator**: A tree view on the left showing project structure with nodes like BOY Examples, CSS, .project, and first.opi.
- Editor**: The central workspace displaying a circular gauge control with numerical scales from 0 to 100.
- Outline**: A panel on the left showing the selected item: Knob(sim://noise).
- Palette**: A list of UI widget types on the right, including Graphics, Monitors, Controls, Scaled Slider, Knob, Scrollbar, Thumb Wheel, Button, Image Boolean Button, Check Box, Combo Box, Radio Box, Choice Button, Others, and Layout.
- Properties**: A detailed properties panel on the right showing settings for the selected Knob component, such as Name (Knob), PV Name (sim://noise), Widget Typ (Knob), and various color and behavior parameters.
- Console**: A bottom panel showing the BOY Console output.



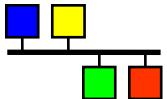
First Try at an OPI

1. Drag ‘knob’ widget from palette to editor
2. Input *sim://noise* as the PV name in Properties view
3. Click the “Run” button



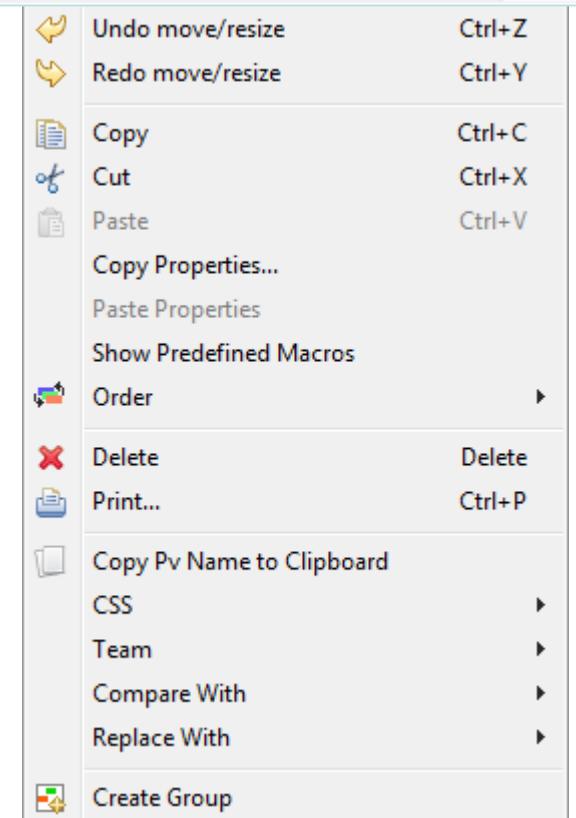
Note: *sim://noise* is a simulated PV inside CSS, so you don't need to run an IOC to try this example. In reality, you can replace it with a real EPICS PV

(See CSS help->CSS Core->Process Variables)



OPI Editing Functions

- What You See Is What You Get (WYSIWYG)
- Comprehensive editing functions on toolbar and context menu
 - Copy/Paste/Delete
 - Drag & Drop
 - Undo/Redo
 - Alignment & Distributing
 - Snap to G (Grid/Geometry/Guide)
 - Zoom In/Out
 - Copy/Paste Properties
 - Changing Orders
 - ...



“As easy as editing PowerPoint”



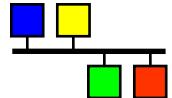
Widget Properties

- Widgets are configured by setting Properties in the *Properties* view
- Common Properties:
 - Name
 - Background/Foreground color
 - Border
 - Font
 - Position
- Widgets that read/write PVs:
 - Basic: [PV Name](#)
 - Border: Alarm Sensitive
 - Behavior: Limits from PV
 - Display: Background/Foreground color
alarm sensitive

Property	Value
Basic	
Name	Knob
PV Name	sim://noise
Widget Type	Knob
Behavior	
Actions	no action
Enabled	<input checked="" type="checkbox"/> yes
Increment	1.0
Level HI	80.0
Level HII	90.0
Level LO	20.0
Level LLO	10.0
Limits From PV	<input checked="" type="checkbox"/> yes
Maximum	100.0
Minimum	0.0
Rules	no rule attached
Scripts	no script attached
Visible	<input checked="" type="checkbox"/> yes
Border	
Alarm Sensitive	<input checked="" type="checkbox"/> yes
Border Color	(0,128,255)
Border Style	None
Border Width	1
Display	
3D Effect	<input checked="" type="checkbox"/> yes
BackColor Alarm Sensit	<input type="checkbox"/> no
Background Color	(240,240,240)
Color HI	(255,255,0)
Color HII	(255,0,0)
Color LO	(255,255,0)
Color LLO	(255,0,0)
Font	Default



Rules



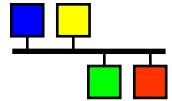
- Rules can be added in the properties.

The screenshot shows the CS Studio interface with the following components:

- Edit Rule Dialog:** A central window titled "Edit Rule" with the following settings:
 - Rule Name: Rule1
 - Property: Background Color (background_color)
 - Boolean Expression: `pv0 < -0.3`
 - Output Value: Major
 - Input PVs:
 - # 0 PV Name: SR01A-PC-VSTR-01:VLOAD Trigger: yes
 - See Generated Script:

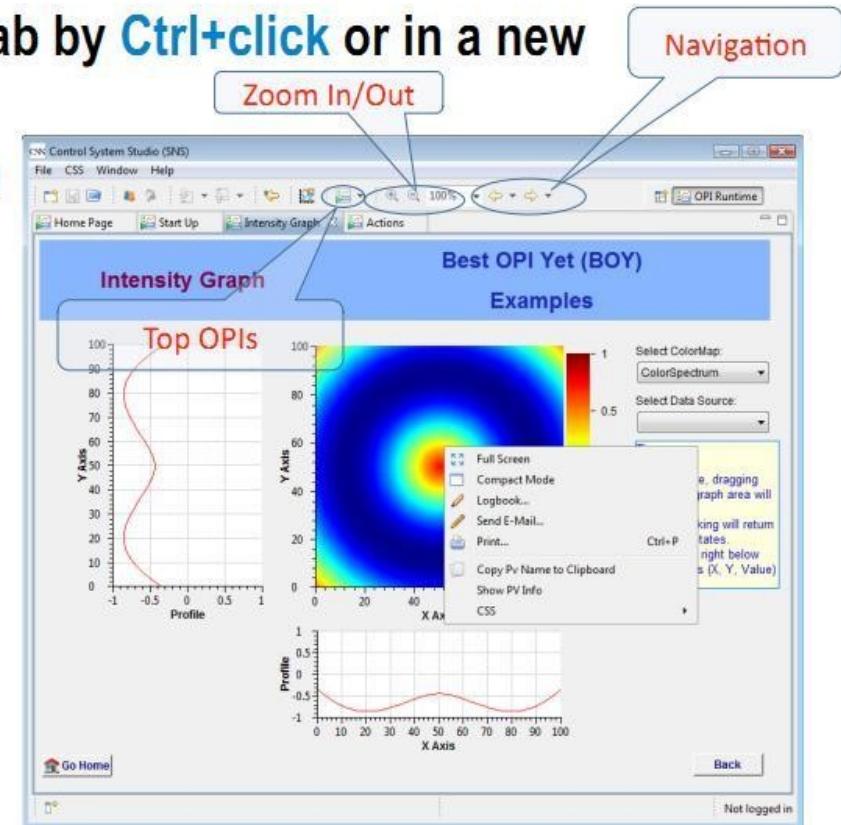
```
importPackage(Packages.org.csstudio.opibuilder.scriptUtil);
var pv0 = PVUtil.getDouble(pvs[0]);
if(pv0 < -0.3)
    widget.setPropertyValue("background_color","Major");
else
    widget.setPropertyValue("background_color","Monitor: BG");
```
- Properties Panel:** A right-hand panel showing properties for the selected rule:

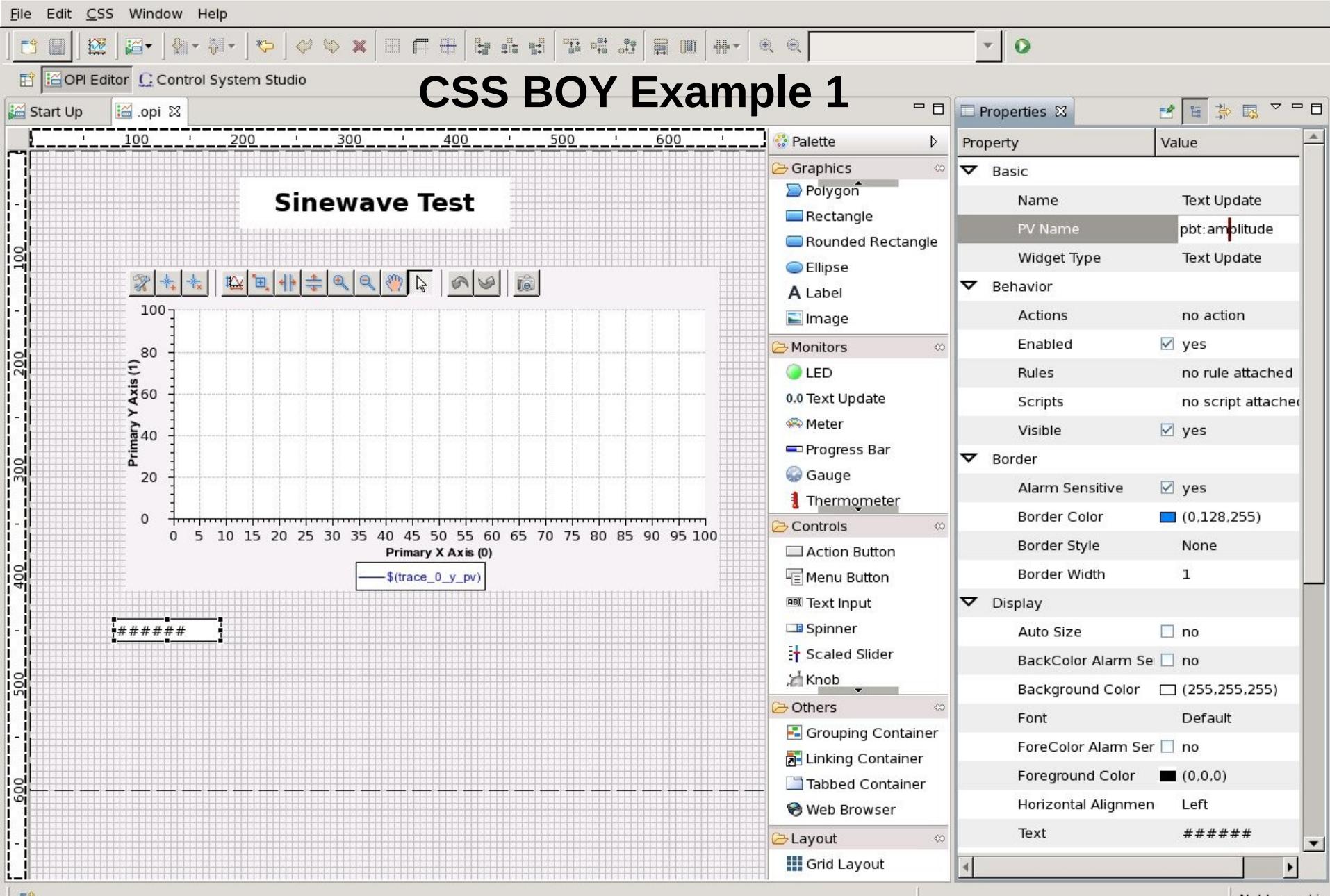
Property	Value
PV Name	SR01A-PC-VSTR-01:VLOAD
Widget Type	Text Update
Behavior	
Actions	no action
Enabled	<input checked="" type="checkbox"/> yes
Rules	no rule attached
Scripts	no script attached
Visible	<input checked="" type="checkbox"/> yes
Wrap Words	<input type="checkbox"/> no
Border	
Alarm Sensitive	<input type="checkbox"/> no
Border Color	Black
Border Style	None
Border Width	0
Display	
Alarm Pulsing	<input type="checkbox"/> no
Auto Size	<input type="checkbox"/> no
BackColor Alarm	<input type="checkbox"/> no
Background Colo	Monitor: BG
Font	Default
Font	Bold
- OPi Editor View:** A bottom-left view showing a PV update for SR01A-PC-VSTR-01:VLOAD with a value of -0.284 V.
- OPi View:** A bottom-right view showing a PV update for SR01A-PC-VSTR-01:VLOAD with a value of -0.316 V, where the background color is red.



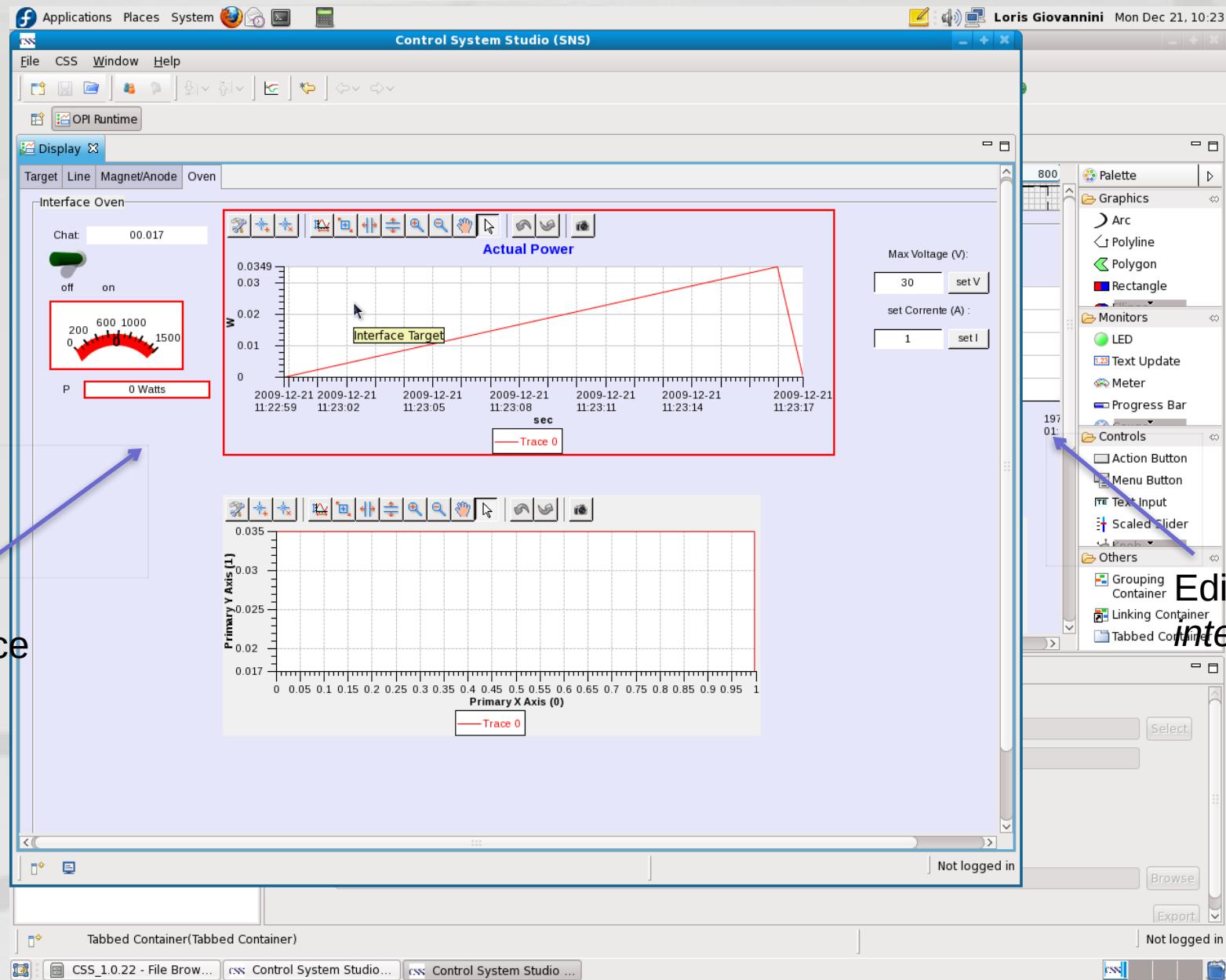
OPI Runtime

- Works like a web browser
 - Display OPIs in Tabs. A tab can be rearranged by dragging it to a new place or new window
 - Open related Display in a new Tab by **Ctrl+click** or in a new Window by **Shift+click**.
 - Backward or Forward Navigation
 - Zoom In/Out
 - Top OPIs
 - Full Screen
 - Compact Mode
 - Elog and Email Screen



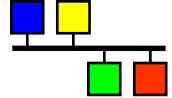


CSS BOY Example 2



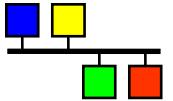
Run
interface

Edit
interface

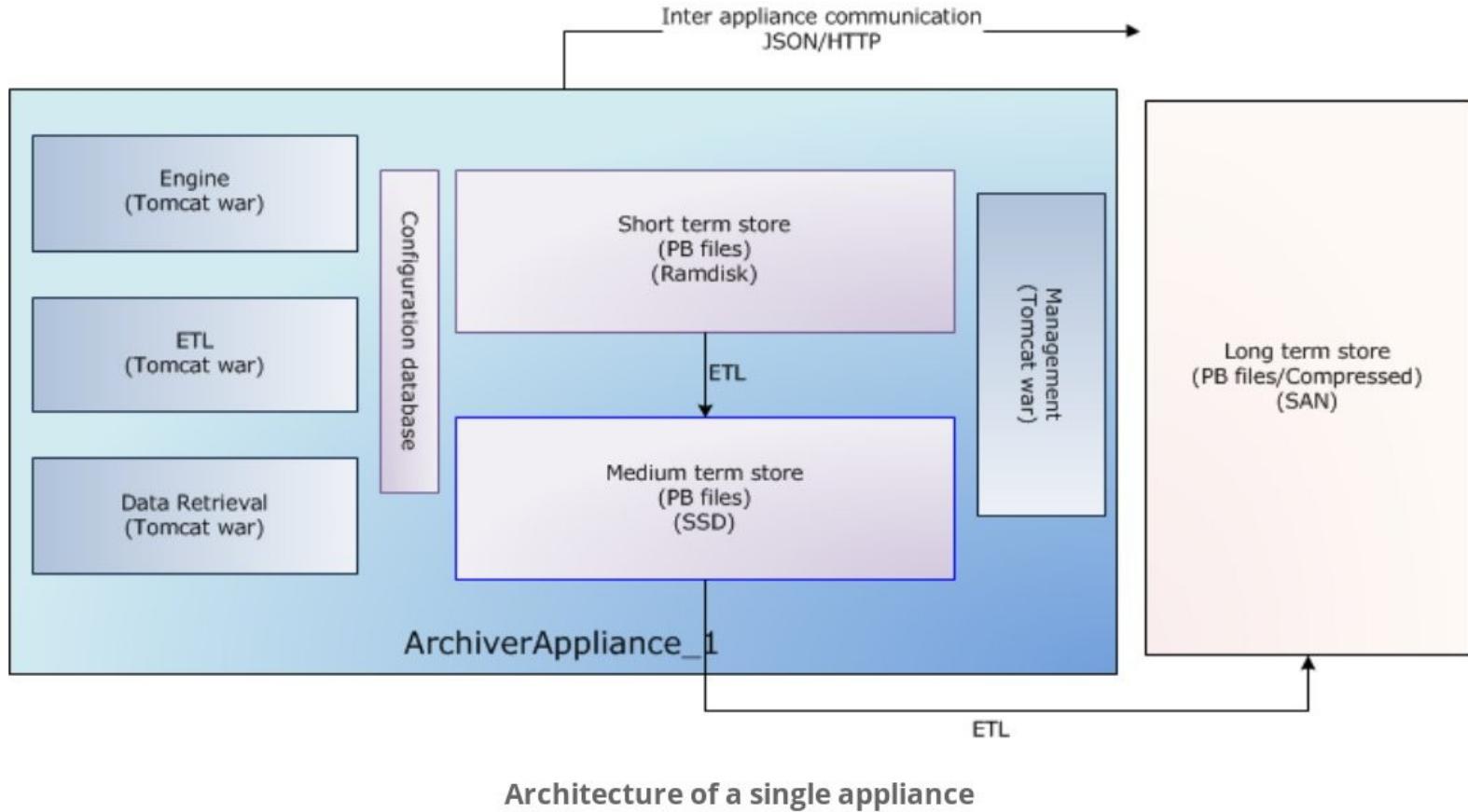


Archiver Appliance

- CA client for automatically saving PVs to disk
- Designed for large scale operations: millions of EPICS PVs, with fast data retrieval
- Multiple storage stages, able to use faster storage where appropriate
- Manage and monitor using Web browser (and RDB)
- Retrieval of data via *CS-Studio* (or the *ArchiveViewer*)
- Limited integration with previous Channel Archiver data sources



Architecture

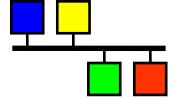


War = Web Application resource SSD = Solid State Drive

ETL = Extract, Transform and Load SAN = Storage Area Network

PB = Google Protocol Buffer

JSON = JavaScript Object Notation

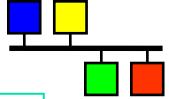


Archiving at Diamond

- At Diamond, there is one instance of the Archiver Appliance handling archiving of all the PVs (currently > 250k) for the accelerator and the beamlines. It can be found here:
<http://archappl.diamond.ac.uk>
- In practice, there are four servers in a cluster behind this URL
- It archives the values of EPICS PVs according to the configuration stored in the relational database.



Archiver Status

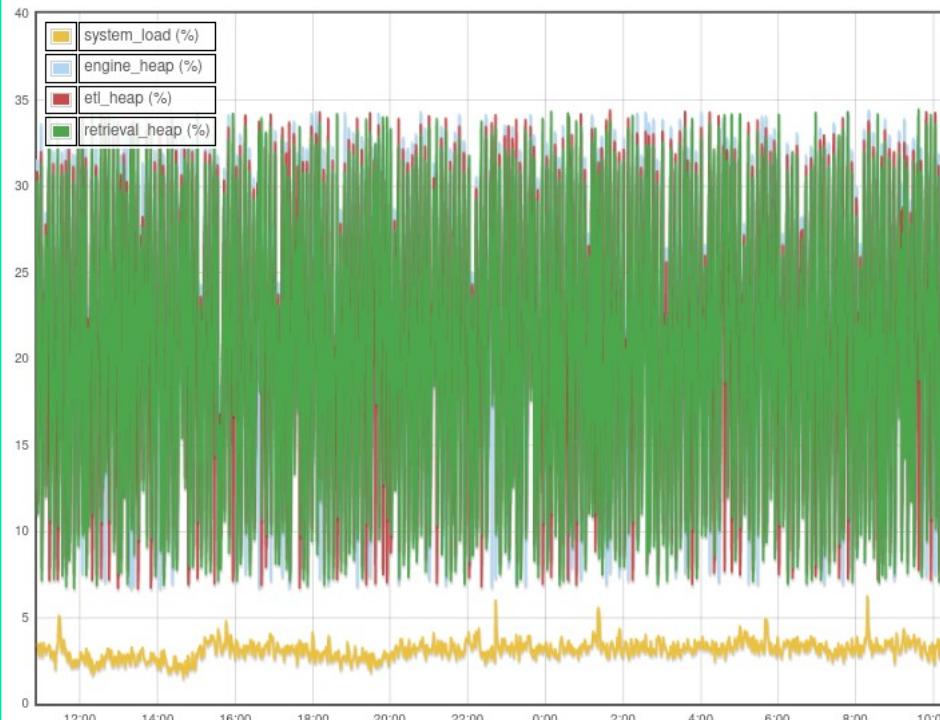


EPICS Archiver Appliance for Diamond Light Source 

Home Reports Metrics Storage Appliances Integration Help

Instance Name	Status	PV Count	Connected	Mgmt Uptime
cs03r-cs-serv-54.pri.diamond.ac.uk	Working	60682	58484	28 days 23:3:39
cs03r-cs-serv-55.pri.diamond.ac.uk	Working	130264	125844	28 days 23:3:39
cs03r-cs-serv-56.pri.diamond.ac.uk	Working	123939	119008	28 days 23:3:40
cs03r-cs-serv-69.pri.diamond.ac.uk	Working	676	676	28 days 23:3:40

system_load (%)
engine_heap (%)
etl_heap (%)
retrieval_heap (%)



Select
Appliances



Archiver Reports

Select Reports

Select

Select

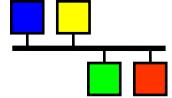
- PVs that may not exist
- Currently disconnected PVs
- Top 100 PVs by event rate
- Top 200 PVs by event rate
- Top 100 PVs by storage rate
- Top 200 PVs by storage rate
- Recently added PVs (100)
- Recently added PVs (200)
- Recently modified PVs (100)
- Recently modified PVs (200)
- PVs by storage consumed (100)
- PVs by storage consumed (200)
- PVs by lost/regained connections (100)
- PVs by lost/regained connections (200)
- PVs by last known timestamp (100)
- PVs by last known timestamp (200)
- PVs by dropped events from incorrect timestamps (100)
- PVs by dropped events from incorrect timestamps (200)
- PVs by dropped events from buffer overflows (100)

EPICS Archiver Appliance for Diamond Light Source

Home Reports Metrics Storage Appliances Integration

Please choose a report: Top 100 PVs by event rate

PV Name	Event Rate	Details	Quick chart
BL15J-DI-QBPM-03:C	50.0998620284757		
BL15J-DI-QBPM-03:B	50.098821757696314		
BL15J-DI-QBPM-03:D	50.098708232699614		
BL15J-DI-QBPM-03:A	50.09846902252575		
BL06I-VA-GAUGE-20:P	10.0		
SR09A-PC-COMPS-01:IDIF	9.996678348634882		
BL23I-MO-DCM-01:FPITCH.RBV	9.933780152967048		
BL23I-MO-DCM-01:FROLL.RBV	9.821817356167667		
BL04I-OP-VFM-01:FPMTR.RBV	9.766831308067887		
BL03I-OP-VFM-01:FPMTR.RBV	9.72810807365676		
SR02A-PC-DTRIM-01:I	9.711088033507705		
BL04I-OP-HFM-01:FPMTR.RBV	9.700379081271869		
BL07I-OP-HFM-01:FPITCH	9.605256030454914		
BL03I-OP-HFM-01:FPMTR.RBV	9.508329283585258		
BL02I-OP-HFM-01:FPMTR.RBV	9.328890630917135		
BL09K-VA-GAUGE-02:PLOG	9.315368826292316		
BL09K-VA-GAUGE-02:P	9.299092173444192		
SR02A-PC-DTRIM-04:I	8.932613514195776		
SR13I-MO-DRIVE-04:INPUT	8.89899602549117		
SR12I-PC-SCMPW-01:I	8.882772207973886		
SR12I-PC-SCMPW-01:DIF	8.882709256236884		
SR02A-PC-S1BE-10:I	8.588533293473011		
SR12I-PC-SCMPW-02:DIF	8.571212403823735		



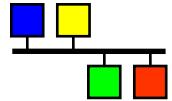
Archiver PV Configuration

- The EPICS archiver is split into different 'archives'. These are managed in the RDB and can be seen by opening

<https://rdb.pri.diamond.ac.uk>

and selecting

EPICS | Archive | Archive Record



Archiving RDB Maintenance

diamond
Philip Taylor

EPICS

- Master Entry
- Device Entry
- Device-DB Information
- Device Location
- Archive
 - Import Archive File
 - Export Archive File
 - Archive Record
 - Data Usage
- Beamline Entry
- Reports

Equipment

Cable

Elog

Detector

Survey & Alignment

Insertion Device

Logout

The wildcard character '%' may be used in search options

Archive Record Maintenance

Archiver Name: All
File Name: All

Record ID:

Retrieve Select the archiver to restart [Restart Archiver](#)

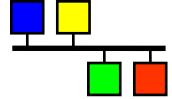
[Records 1 to 10 of 282356] [Page 1 of 28236]

SI No.	Select All	Archiver Name	Record ID	Archive Freq	Archive Type	Updated by File Name	
1	<input type="checkbox"/>	I04-1	0:CA_CLNT_CNT	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
2	<input type="checkbox"/>	I04-1	0:CA_CONN_CNT	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
3	<input type="checkbox"/>	I04-1	0:FD_FREE	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
4	<input type="checkbox"/>	I04-1	0:IOC_CPU_LOAD	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
5	<input type="checkbox"/>	I04-1	0:MEM_FREE	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
6	<input type="checkbox"/>	I04-1	0:MEM_USED	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
7	<input type="checkbox"/>	I04-1	0:SUSP_TASK_CNT	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
8	<input type="checkbox"/>	I04-1	0:SYS_CPU_LOAD	AA SB-AA	1	Monitor	James O'Hea 24/07/2018 16:34:00 /dls_sw/work/R3.14.12.3/loc/BL04J/BL/data/BL04J.grp
9	<input type="checkbox"/>	R79-VA	BK-VA-BKOUT-01:GETSTA	AA SB-AA	.1	Monitor	K.Vijayan 03/11/2017 17:41:10 /home/kv73/Downloads/R79-VA.grp Manual Update
10	<input type="checkbox"/>	R79-VA	BK-VA-BKOUT-01:MAXT	AA SB-AA	10	Monitor	K.Vijayan 30/05/2018 17:22:07 /home/kv73/Downloads/R79-VA.grp Manual Update K.Vijayan 11/06/2018 20:09:41

Select Records per page: 10

[Previous 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48]
49 50... of 28236 [Next]

Save Delete Print Reset



Sampling Methods

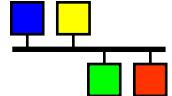
Sampling Methods

There are two ways to specify when PV data is archived. In each case, a period must also be specified:

- **Scan**: sample the PV value at a fixed period using CA get
- **Monitor**: set up a CA monitor and record the value as each monitor occurs

The ADEL field

PVs often have an .ADEL field. The value of this field is known as the *archive deadband*. This is the amount the PV value needs to change in order that the archiver registers an update. This is used to avoid excessive archiving of noisy signals.



Group Files

PV configurations are imported and exported using group files. The general format is:

PV-NAME <number> <method>

Where

<number> may be integer or floating point (seconds).

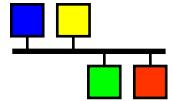
<method> may be SCAN or MONITOR.

- SCAN means that the archiver will record a sample at the specified interval.
- MONITOR means that every monitored PV change is recorded; the number determines the buffers reserved for the data by the archiver e.g. 10 means allocate buffers for one value every 10 seconds

A PV may not appear twice in a group file. Here is an example group file:

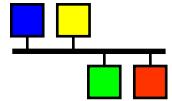
BR-RF-CRCAV-01:PF-S 1.4 Monitor

BL16B-OP-SLITS-03:Y:MINUS:ELOSS.VAL 1 Scan



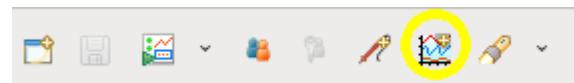
Viewing data using CSS

- The CSS Data Browser replaces the functionality of Strip Tool and Channel Archive Viewer in a single application.
- It can graph live PV data and Archived PV data in the same plot.
- The data browser is an integral part of Control System Studio (CSS), however it can be used with any EPICS PVs so can be used alongside the existing EDM panels at Diamond.



Using CSS Data Browser

1. Open the Launcher and select Utilities → Control Systems Studio
2. CSS should start and restore your most recent layout (if any)
3. Click the button 'Open new Data Browser plot' in the main toolbar:

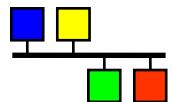


4. You might need to right click on the perspective button labelled 'Data Browser' in the top right and choose 'Reset'.
5. The Archive Search tab on the left hand pane allows discovery of those PVs available in the Archiver, using wildcard PV specifiers e.g.

PV Name	Name
BL16I-MO-DIFF-01:BASE:Y1.DMOV	All-ArchiveEngine
BL16I-MO-DIFF-01:BASE:Y2.DMOV	All-ArchiveEngine
BL16I-MO-DIFF-01:BASE:Y3.DMOV	All-ArchiveEngine
BL16I-MO-DIFF-01:SAMPLE:MU.DMOV	All-ArchiveEngine



Using CSS Data Browser



CS-Studio

File Edit Search CS-Studio Window Help

Archive ... Navigator Standal... Data Browser

URL: Primary Archiver Appliance - pbraw Info

Name Description Key

EPICS Archiver Appli http://epicsarchiver:1

Pattern: BL16I+?:?DMOV Search

Add... Replace search results Reg.Exp.

PV Name	Name
BL16I-MO-DIFF-01:BASE:Y1	EPICS Archiver Appli
BL16I-MO-DIFF-01:SAMPLE:	EPICS Archiver Appli
BL16I-MO-DIFF-01:BASE:Y3	EPICS Archiver Appli
BL16I-MO-DIFF-01:BASE:Y2	EPICS Archiver Appli

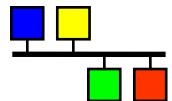
Properties Export Samples

Property Value

bvf81224



Using CSS Data Browser



Plotting Search Results

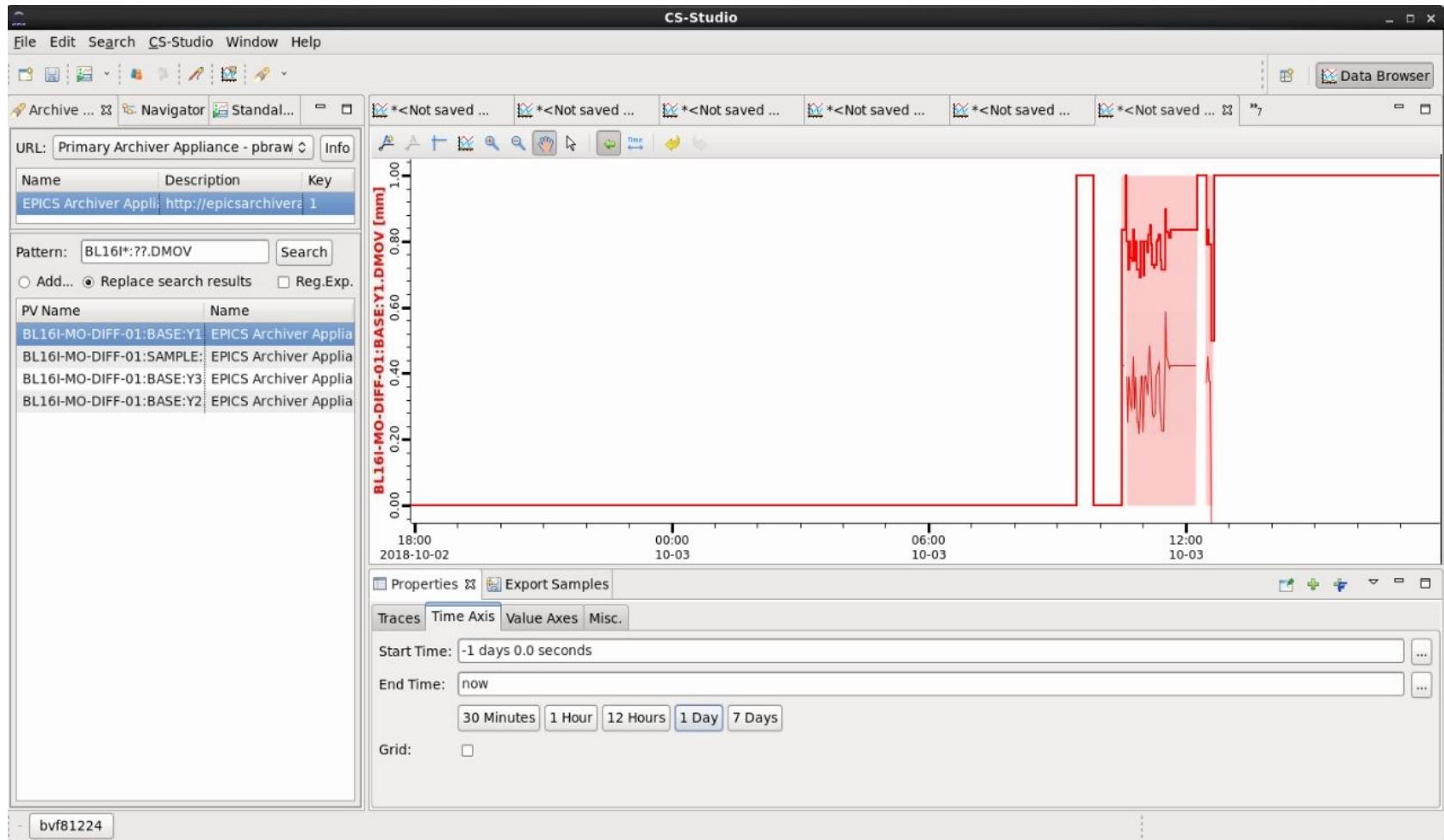
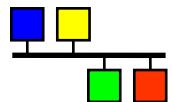
Once you have a list of PVs in the Archive Search Results you have the following choices:

1. Double click a PV name. This opens a new Data Browser Plot window containing just this one PV
2. Drag the PV name into an existing Plot Window

This adds the PV to those already being plotted. You will be given a choice of creating a new vertical axis for this PV or adding it to an existing vertical axis

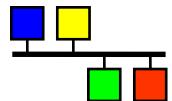


Using CSS Data Browser





Data Browser Options

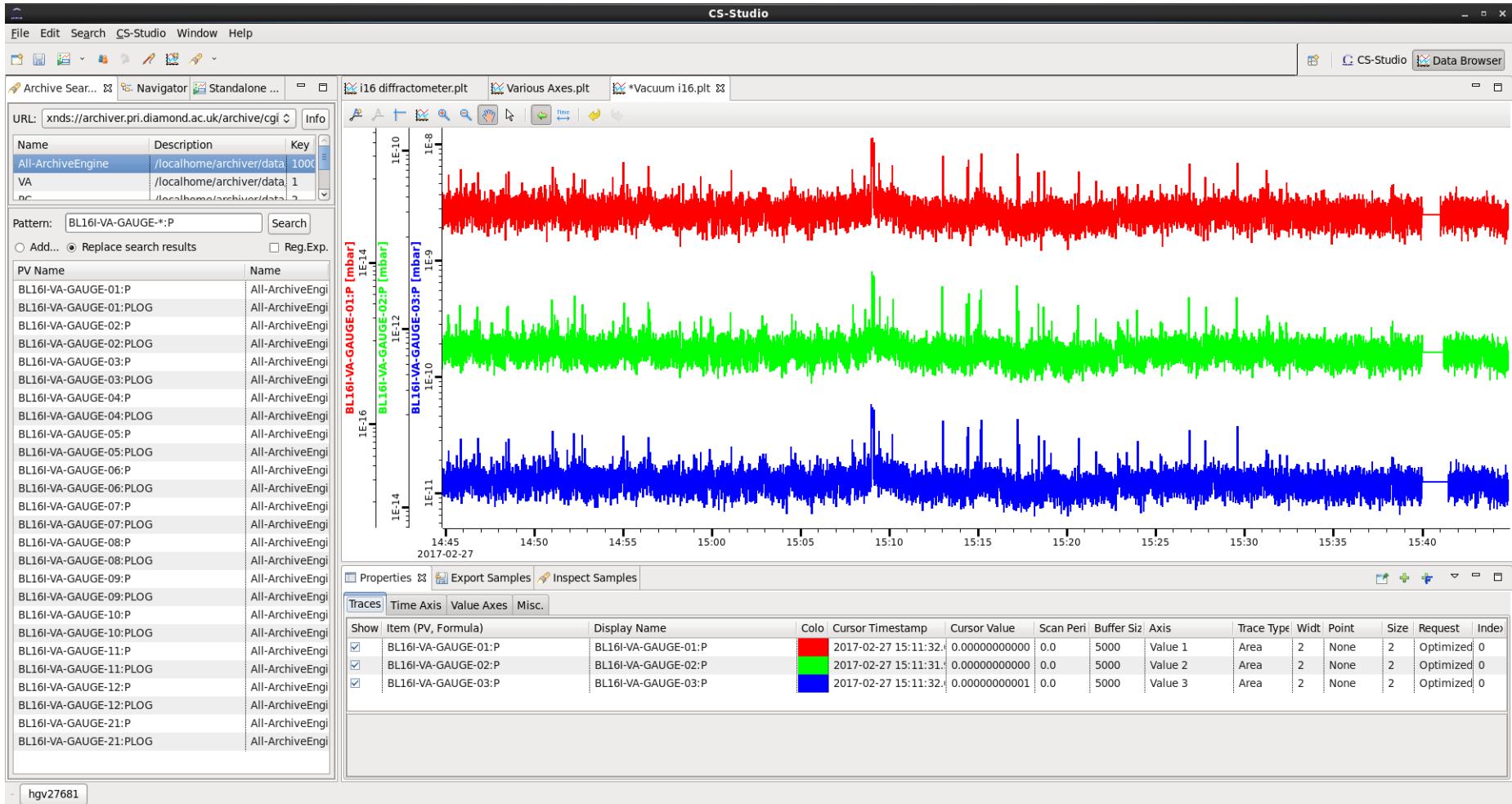
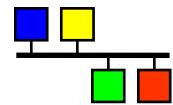


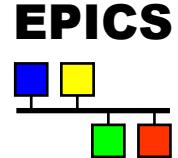
Change trace properties

Change time frame



Using CSS Data Browser





asynPortDriver

C++ Base Class for asyn Port Drivers

Andy Foster
Observatory Sciences Limited

(based on a talk by Mark Rivers)

asynPortDriver

- “New“ (2009) C++ base class that greatly simplifies writing an asyn port driver
 - Initially developed as part of the areaDetector module
 - All areaDetector drivers use this
 - Recently motor drivers, now use asynPortDriver
- Hides all details of registering interfaces, doing callbacks, connection management
- Why C++? Things that are hard in C:
 - Inheritance:
 - virtual base class functions that can be overridden or enhanced by derived classes
 - Template functions: single function can handle any data type.
 - Used extensively in areaDetector which supports 8 data types for NDArrays

asynPortDriver C++ Base Class

- Parameter library
 - Drivers typically need to support a number of parameters that control their operation and provide status information. Most of these can be treated as integers, binary values, floats or strings. Sequence for new value:
 - New parameter value arrives in driver from EPICS output record, or new data arrives from device
 - Change values of one or more parameters
 - For each parameter whose value changes set a flag noting that it changed
 - When operation is complete, force callbacks (to update records with SCAN set to “I/O Intr”)
- asynPortDriver provides methods to simplify the above sequence

asynPortDriver C++ Parameter Library Methods

```
virtual asynStatus createParam( const char *name, asynParamType type, int *index);

virtual asynStatus setIntegerParam( int addr, int index, int value );
virtual asynStatus setDoubleParam( int addr, int index, double value );
virtual asynStatus setStringParam( int addr, int index, const char *value );

virtual asynStatus getIntegerParam( int addr, int index, int *value );
virtual asynStatus getDoubleParam( int addr, int index, double *value );
virtual asynStatus getStringParam( int addr, int index, int maxChars, const char
*value );

virtual asynStatus callParamCallbacks( int addr );
```

- *createParam* takes a name string and creates a numbered parameter with associated data type
- drivers use *set* and *get* methods to write and read values to/from the parameter library
- There is a separate parameter library for each asyn “address” that the driver supports e.g. each axis on a motion controller
- The parameter library maintains a changed flag for each parameter
- *callParamCallbacks* updates EPICS records (SCAN=“I/O Intr”) when the parameter they are interested in has changed value

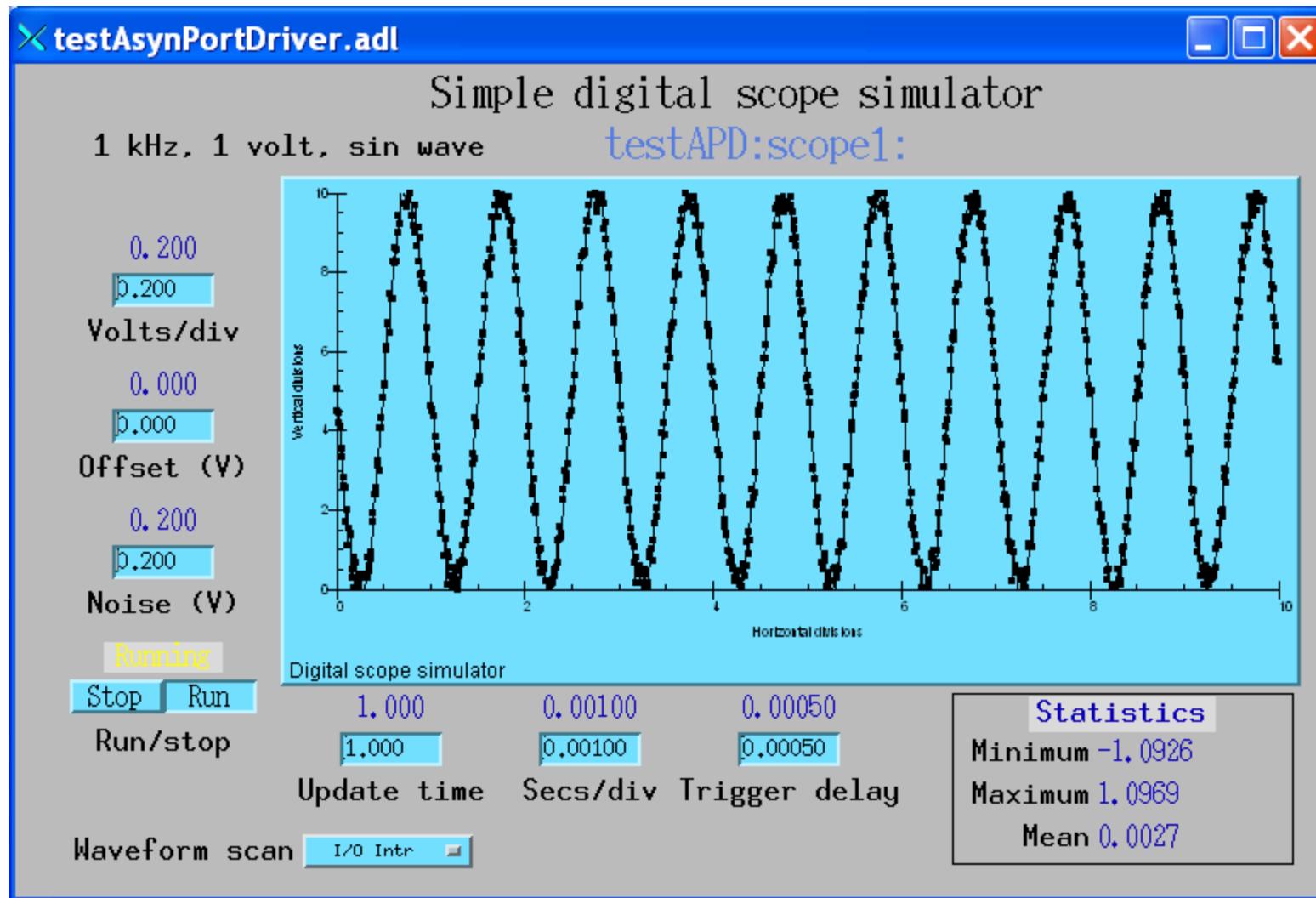
asynPortDriver Write/Read Methods

```
virtual asynStatus readInt32(asynUser *pasynUser, epicsInt32 *value);
virtual asynStatus writeInt32(asynUser *pasynUser, epicsInt32 value);
virtual asynStatus readFloat64(asynUser *pasynUser, epicsFloat64 *value);
virtual asynStatus writeFloat64(asynUser *pasynUser, epicsFloat64 value);
virtual asynStatus readOctet(asynUser *pasynUser, char *value, size_t maxChars,
                           size_t *nActual, int *eomReason);
virtual asynStatus writeOctet(asynUser *pasynUser, const char *value,
                           size_t maxChars, size_t *nActual);
virtual asynStatus readInt16Array(asynUser *pasynUser, epicsInt16 *value,
                                 size_t nElements, size_t *nIn);
virtual asynStatus writeInt16Array(asynUser *pasynUser, epicsInt16 *value,
                                 size_t nElements);
```

- When an EPICS input record processes, a request is made to access the port. When the port is free, the base class readXXX function is called, which gets the value from the parameter library.
- Very rarely need to override readXXX functions
- When an EPICS output record processes, a request is made to access the port. When the port is free, the base class writeXXX function is called. By default, this sets the value in the parameter library. Override this to add code to access the hardware i.e. send command to hardware.
- EPICS input records on “I/O Intr” scan receive their updates via callbacks from the driver, as a result of *callParamCallbacks* being called.

testAsynPortDriver

Digital Oscilloscope Simulator



testAsynPortDriver

Digital Oscilloscope Simulator

- 18 records (ao, ai, bo, bi, longin, waveform)
- All input records are "*I/O Intr*" scanned
- Only 430 lines of well-commented C++ code
- Look in: /dls_sw/prod/R3.14.12.7/support/asyn/4-41/testAsynPortDriverApp/src

testAsynPortDriver.h – Class definition

```
class testAsynPortDriver : public asynPortDriver {  
public:  
    testAsynPortDriver(const char *portname, int maxArraySize);  
  
    // These are the methods that we override from asynPortDriver  
    virtual asynStatus writeInt32(asynUser *pasynUser, epicsInt32 value);  
    virtual asynStatus writeFloat64(asynUser *pasynUser, epicsFloat64 value);  
  
    void simTask(void);  
  
protected:  
    // Indexes into the parameter library  
    int P_VertGain;  
    int P_UpdateTime;  
    int P_Run;  
  
private:  
    // Our data and methods  
    void setVertGain();  
};  
#define NUM_SCOPE_PARAMS 3
```

Relationship between Parameters and EPICS Records

```
// These are the drvInfo strings that are used to identify the parameters
#define P_VertGainString      "SCOPE_VERT_GAIN_SELECT" /* asynInt32 */
#define P_UpdateTimeString    "SCOPE_UPDATE_TIME"        /* asynFloat64 */
#define P_RunString           "SCOPE_RUN"                 /* asynInt32 */

// In the constructor, we do this
createParam(P_VertGainString,      asynParamInt32,   &P_VertGain);
createParam(P_UpdateTimeString,    asynParamFloat64, &P_UpdateTime);
createParam(P_RunString,          asynParamInt32,   &P_Run);

// In the database, we have this
record(mbbo, "$(P)$(R)VertGainSelect") {
    field(PINI, "1")
    field(DTYP, "asynInt32")
    field(OUT, "@asyn($(PORT),$(ADDR),$(TIMEOUT))SCOPE_VERT_GAIN_SELECT")
    field(ZRST, "1")
    field(ZRVL, "1")
    field(ONST, "10")
    field(ONVL, "10")
}
record(ao, "$(P)$(R)UpdateTime") {
    field(PINI, "1")
    field(DTYP, "asynFloat64")
    field(OUT, "@asyn($(PORT),$(ADDR),$(TIMEOUT))SCOPE_UPDATE_TIME")
    field(PREC, "3")
}
```

There are also Input Records!

```
record(ai, "$(P)$RVertGainSelect_RBV")
{
    field(PINI, "1")
    field(DTYP, "asynInt32")
    field(INP, "@asyn($(PORT),$(ADDR),$(TIMEOUT))SCOPE_VERT_GAIN")
    field(PREC, "0")
    field(SCAN, "I/O Intr")
}

record(ai, "$(P)$RUpdateTime_RBV")
{
    field(PINI, "1")
    field(DTYP, "asynFloat64")
    field(INP, "@asyn($(PORT),$(ADDR),$(TIMEOUT))SCOPE_UPDATE_TIME")
    field(PREC, "3")
    field(SCAN, "I/O Intr")
}
```

- Input and output records so that if the driver is forced to modify a parameter (e.g. because the device cannot support the output value it received or changing this parameter affects another one) there is feedback to the user on the actual values being used.

testAsynPortDriver Constructor

```
testAsynPortDriver::testAsynPortDriver(const char *portName, int maxPoints)
: asynPortDriver(portName,
                  1, // maxAddr
                  (int)NUM_SCOPE_PARAMS,
asynInt32Mask|asynFloat64Mask|asynFloat64ArrayMask|asynEnumMask|asynDrvUserMask,
                                         // Interface mask
asynInt32Mask | asynFloat64Mask | asynFloat64ArrayMask | asynEnumMask,
                                         // Interrupt mask
                  0, // non-blocking driver
                  1, // autoconnect
                  0, // default priority for port thread
                  0) // default stack size for port thread
createParam(P_VertGainString,      asynParamInt32,    &P_VertGain);
createParam(P_UpdateTimeString,   asynParamFloat64,  &P_UpdateTime);
createParam(P_RunString,          asynParamInt32,    &P_Run);

// Set the initial values of some parameters
setIntegerParam(P_VertGainSelect, 10);
setVertGain(10);
setDoubleParam(P_UpdateTime, 0.5);
setIntegerParam(P_Run, 0);

// Create the thread that computes the waveforms in the background

status = (asynStatus)(epicsThreadCreate("testAsynPortDriverTask",
                                         epicsThreadPriorityMedium,epicsThreadGetStackSize(epicsThreadStackMedium),
                                         (EPICSTHREADFUNC)::simTask, this) == NULL);
```

testAsynPortDriver – IOC startup

- How is the constructor called?
- In the driver we create a function:

```
int testAsynPortDriverConfigure(const char *portName, int maxPoints)
{
    new testAsynPortDriver(portName, maxPoints);
    return(asynSuccess);
}
```

- In the IOC startup script, we have the line:

```
testAsynPortDriverConfigure("testAPD", 1000)
```

testAsynPortDriver – override writeInt32

```
asynStatus testAsynPortDriver::writeInt32(asynUser *pasynUser,
epicsInt32 value)
{
    int index          = pasynUser->reason;      // index into parameter library
    asynStatus status = asynSuccess;

    /* Set the value of this parameter in the parameter library */
    status = (asynStatus) setIntegerParam(index, value);

    if (index == P_Run) {           // parameter which controls running the scope
        if (value) epicsEventSignal(this->eventId); // wake up simulation task
    }
    else {
        // All other parameters just get set in parameter list, no need to act on them here
    }

    /* Do callbacks so EPICS Records on "I/O Intr" see any changes */
    status = (asynStatus) callParamCallbacks();
    return status;
}
```

testAsynPortDriver – override writeFloat64

```
asynStatus testAsynPortDriver::writeFloat64(asynUser *pasynUser, epicsFloat64
value){
    int          index  = pasynUser->reason;           // Index into parameter library
    asynStatus status = asynSuccess;

    /* Set the parameter in the parameter library */
    status = (asynStatus) setDoubleParam(index, value);

    if (index == P_UpdateTime)
    {
        // Make sure the update time is valid. If not change it and put back in parameter library
        if (value < MIN_UPDATE_TIME)
        {
            asynPrint(pasynUser, ASYN_TRACE_WARNING,
                      "%s: warning, update time too small, changed from %f to %f\n",
                      functionName, value, MIN_UPDATE_TIME);
            value = MIN_UPDATE_TIME;
            setDoubleParam(index, value);
        }
    } else {
        // All other parameters just get set in parameter list, no need to act on them here
    }
    // Do callbacks so EPICS records see any changes
    status = (asynStatus) callParamCallbacks();

    return status;
}
```

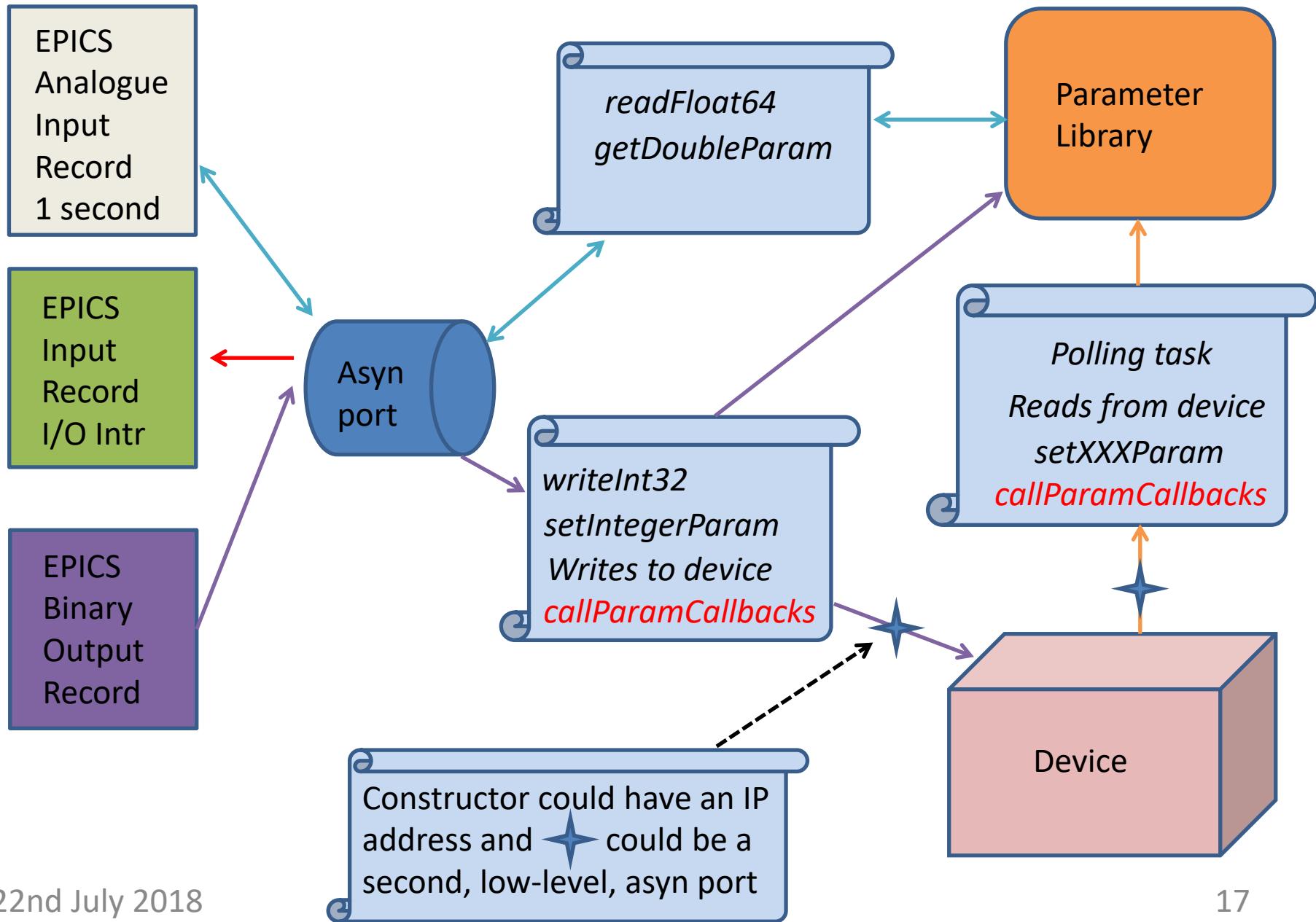
testAsynPortDriver – simulation task

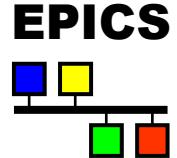
```
void testAsynPortDriver::simTask(void)
    // This separate thread computes the waveform and does callbacks with it
{
    /* Loop forever */
    lock(); // we need to lock when this tasks wants access to the parameter library!
    while (1) {
        getDoubleParam(P_UpdateTime, &updateTime);
        getIntegerParam(P_Run, &run);
        unlock(); // we need to unlock here in case we get held up below waiting...
        if (run)
            epicsEventWaitWithTimeout(this->eventId, updateTime);
        else
            epicsEventWait(this->eventId);
        lock();
        getIntegerParam(P_Run, &run); // run could have changed while we were waiting
        if (!run)
            continue;
        else
        {
            // Access parameter library and use values to calculate new waveform
            ...
            callParamCallbacks();
            doCallbacksFloat64Array(pData_, maxPoints, P_Waveform, 0);
                // special call to update waveform records
        }
    }
}
```

testAsynPortDriver – Real Device Control?

- How would we use this driver as a basis for real device control?
- Change simulation task to talk to an actual device (poller task reads from hardware, puts values into parameter library; I/O Intr scanned records updated from library via callbacks.
- writeXXX functions access hardware. Set value into hardware, check that it wrote (i.e. read back), put value into parameter library. If write failed, put current device value into library.
- Note that, if using asyn at this lower (hardware) level, we can use the asynXXXSyncIO interface, which does a write followed by a read.

testAsynPortDriver – Real Device Control?

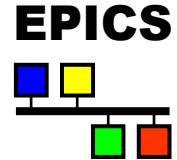




Custom Records

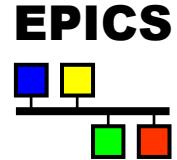
Andy Foster

Observatory Sciences Limited



Outline

- The genSub record
- The aSub record
 - Differences between aSub and genSub
 - aSub/genSub verses custom records
- The EPID record
- The Transform record

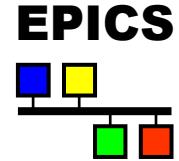


genSub record

- The genSub (General Subroutine) record was written in 1996 as an enhancement to the existing EPICS sub record.
- Requirements were:
 - Handling of multiple inputs and outputs
 - Wanted a way of passing arrays and strings between systems
- Up to 21 input and 21 output fields
- Each field type configurable by user
- Handles structures
- Routine called at process time can be changed dynamically
- Event posting for changes in output field values is configurable
- Never formally added to EPICS base... but widely used!



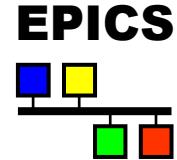
Input and Output fields: common properties



- INPA - Input link to 'A'
- FTA - Type of 'A'
- NOA - Number of elements in 'A'
- UFA - User function 'A' (for structures)



Example database fragment

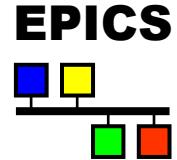


- To use the “A” field to read an array from another record, you would need a database fragment like this:

```
record(genSub, "ioc1:genSub1") {  
    field(SNAM, "myProcess")  
    field(FTA,      "LONG")  
    field(NOA,      "100")  
    field(INPA,     "ioc1:waveform1") ... }
```



Example subroutine fragment



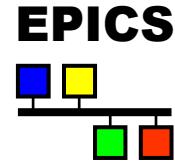
- The associated subroutine code that uses the “A” field might look like this:

```
static long myProcess(genSubRecord *prec)
{
    long i, *a;
    double sum=0;
    a = (long *)prec->a;
    for( i=0; i<prec->noa; i++ )
        sum += a[i];           // Sum the elements of the array in the “A” field
    return(0)
}
```

- Note that the subroutine must always treat value fields (A-U, VALA-VALU) as pointers



User defined structures



- To pass a structure between 2 “genSub’s”

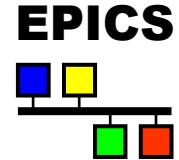
```
struct pinfo
{
    int      age;
    char    name[128];
    double   salary;
};
```

- User supplies a function which returns size. This is specified in the “UF<n>” field:

```
long setup( struct genSubRecord *pgsub )
{ return( sizeof(struct pinfo) ); }
```



User defined structures contd.



- To pass this structure between output field VALA of record 1 and input field A of record 2, the following property settings would be required:

Record 1:

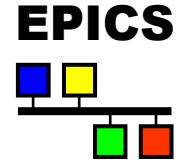
UFVA: **setup** // function name
FTVA: CHAR
NOVA: 1

Record 2:

UFA: **setup** // function name
FTA: CHAR
NOA: 1



User defined structures contd.



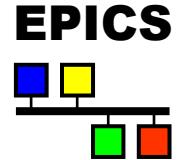
- Inside the user routine called (at process time) from record 2, the elements of the structure become available to the routine by the cast:

```
struct pinfo *data;  
data = (struct pinfo *)pgsub->a
```

- Has side effect of allowing a character string greater than 40 characters to be passed over Channel Access.

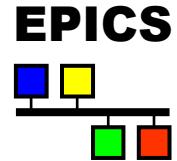


Dynamic change of processing routine



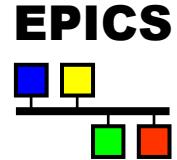
Two ways to do this:

- Set LFLG = IGNORE and change SNAM with a “ca_put”
- Or if LFLG = READ, the name of routine is read from the SUBL link field:
 - Record looks up address of new routine
 - During processing, new routine is called
 - Routine needs to have been loaded in the IOC at iocInit



Other things to note...

- The genSub record does not talk to device support
 - If you want to write to a device, you would use your output fields and links to write to standard records that can write to hardware
- Posting of output value changes is determined by the EFLG (Event Flag) setting:
 - Set to NEVER or ALWAYS or ON CHANGE

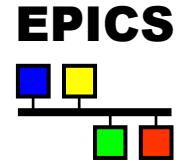


aSub record

- Array Subroutine record added to Base at EPICS 3.14.10 (October 2008).
- Heavily based on the *genSub* record.
- Like the *genSub*, the *aSub* does not talk to device support.



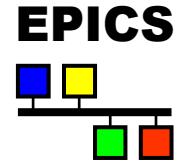
Differences between the genSub and the aSub



- The fields UFA..UFU, UFVA..UFVU have been removed.
 - Hence, no support for passing structures
- The default value for the FTA..FTU and FTVA..FTVU fields has been changed from "STRING" to "DOUBLE", to reduce the memory footprint of unused fields.



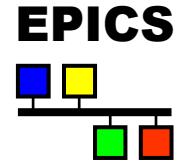
Differences between the genSub and the aSub



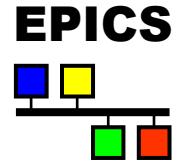
- For aSub records the value returned by the subroutine is an error status
 - if return < 0 then an alarm is raised with status SOFT_ALARM and severity controlled by the BRSV field.
 - **If return > 0 no alarm but no OUTx links processed!**
 - **The OUTx links only process if the subroutine return value is 0!!**



aSub and genSub records verses custom records

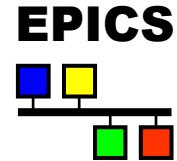


- When would you use an aSub or genSub record rather than using a custom record for a particular task?
- The answer depends on:
 - Does the custom record exist?
 - Does it have enough input and output fields?
- The disadvantages of using aSub/genSub are:
 - Bigger memory footprint (especially for embedded IOC's)
 - More danger of crashing the IOC due to a coding error
- The major advantage of aSub/genSub is the total flexibility to do whatever you want in “C” code and have multiple inputs and outputs.
- Typical applications for aSub/genSub records include FFT's, LUT's and telescope pointing kernels.



EPID record

- The Enhanced PID record is used to implement closed-loop control within an EPICS database.
 - e.g. feedback loop between a beam position (vertical) monitor and the fine pitch control of a DCM
- There are three fields in the record representing the PID gains as follows:
 - Proportional gain KP
 - Integral gain, KI
 - Derivative gain, KD
- Using these gains, the user can “tune” the control loop according to the characteristics of the system.

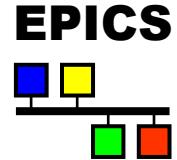


EPID: Soft Device Support

- Input link (INP) reads the value of the controlled process variable into the CVAL field.
- It is good if INP is a database link (same IOC), as this cuts down latency, but it does not have to be.
- Device support calculates PID expression and, if FBON=1, writes OVAL to the output link OUTL.
- The SCAN field controls the rate at which the feedback calculation is actually performed. The maximum rate is typically 20 Hz, although it could be run faster (<100Hz).



Feedback Parameters



The discrete form of the PID expression is as follows:

$$M(n) = P + I + D$$

$$P = KP * E(n)$$

$$I = KP * KI * \text{SUM}_i (E(i) * dT(n))$$

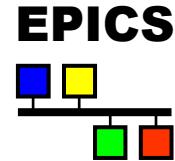
$$D = KP * KD * ((E(n) - (E(n-1)) / dT(n))$$

where

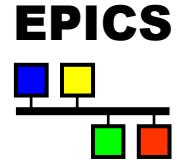
- $M(n)$ = value of manipulated variable at n th instant.
- P = Proportional term
- I = Integral term
- D = Derivative term
- KP = Proportional gain
- KI = Integral gain
- KD = Derivative gain
- $E(n)$ = Error at n th sampling instant
- SUM_i = Sum from $i=0$ to $i=n$
- $dT(n)$ = Time difference between $n-1$ instance and n th instance
- The terms KP , KI , and KD are the only terms configured by the user.



Feedback Parameters



- $\text{ERR} = (\text{VAL}-\text{CVAL})$ (setpoint – controlled variable)
 - ERR is the error at the nth sampling instant
- Note that the integral term, I, is the sum from time=0 to the present time
 - This can grow extremely large if not subject to some "sanity checks". Original PID record did not do this!
 - The EPID record does cap the value of I



Transform Record

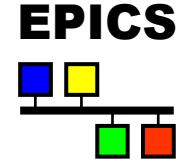
- This record is used to perform bi-directional coordinate transformations. Examples are:
 - A horizontal mirror supported underneath with 3 Y-jacks going between (Y1, Y2, Y3) and (Pitch, Roll, Height)
 - Slits going between positive/negative blade position control and gap/centre control
- Nowadays these sort of transformations are more often done in the PMAC (Geobrick) using kinematics

EPICS Sequencer and State Notation Language (SNL)

Original: Ned Arnold (APS)

Updates by Philip Taylor and Emma Arandjelovic (OSL)

State Notation Compiler and Sequencer



- Allows programming of sequential state-oriented operations to run in the IOC
- The program interacts with the run-time database(s) via channel access
- Home page for snc/seq is:
<http://www-csr.bessy.de/control/SoftDist/sequencer/>
Latest stable release is V2.2.7
- Recent versions of snc/seq (V2.0 onwards, with EPICS 3.14 base) can run on any supported EPICS target *or host*.
- Pre V2.0 could only run under vxWorks.

Advantages

- Can implement complicated algorithms
- Can stop, reload, restart a sequence program without rebooting
- Interact with the operator through string records and mbbo records
- C code can be embedded as part of the sequence
- File access can be implemented quite easily
- All Channel Access details are taken care of for you

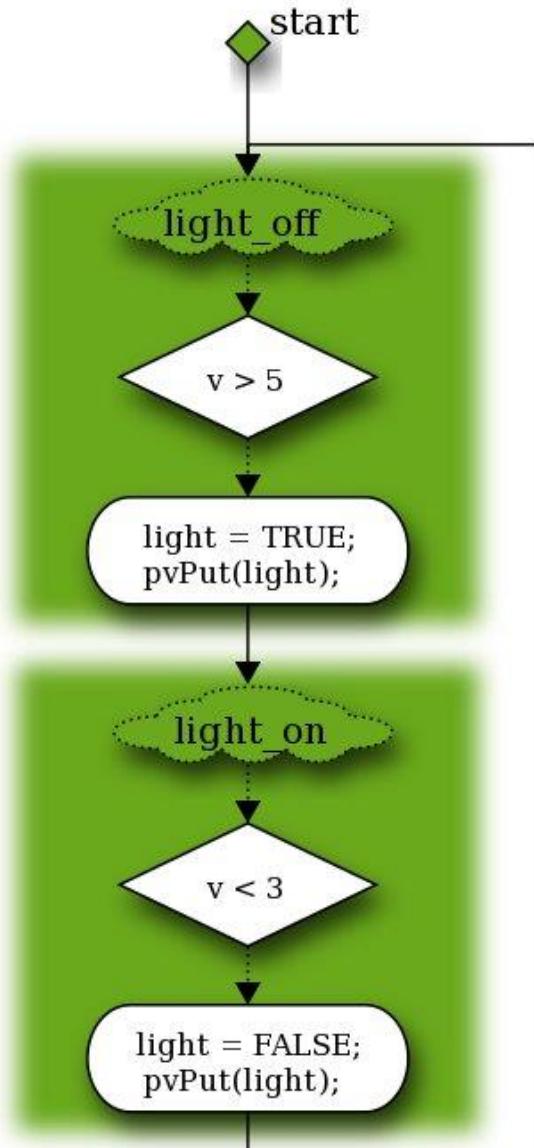
Definitions

- *SNL* : State Notation Language
- *SNC* : State Notation Compiler
- *sequencer* : The tool within the IOC that executes the compiled SNL code
- *Program* : A complete SNL application consisting of declarations and one or more state sets
- *State Set* : A set of states that make a complete finite state machine
- *State* : A particular mode of the state set in which it remains until one of its transition conditions is evaluated to be TRUE

State Notation Language SNL

- A domain specific language for programming control applications in co-operation with an EPICS database
- Typically used when control flow gets more complex than can be easily achieved with records and links
- SNL is a (small) subset of the C language, plus features for:
 - specifying finite state machines (state sets, keyword `ss`)
 - associating program variables with PVs (`assign`)
 - automatically updating their value (`monitor`)
 - channel Access operations with PVs (`pvPut`, `pvGet`)
- SNL compiler translates SNL to C

Example



In state `light_off`, whenever the voltage (`v`) exceeds a value of 5.0, the light switch is turned on, and the internal state changes to `light_on`.

In state `light_on`, whenever the voltage drops below 3.0, the light switch is turned off, and the internal state changes to `light_off`.

Example SNL

The SNL code structure follows the state transition diagram format

```
ss volt_check {
    state light_off {
        when (voltage > 5.0) {
            light = TRUE;
            pvPut(light);
        } state light_on
    }
    state light_on {
        when (voltage < 3.0) {
            light = FALSE;
            pvPut(light);
        } state light_off
    }
}
```

SNL Basics

- Each state has one or more when statements which specify which state to enter next if their condition is met
- Action statements are executed during transitions

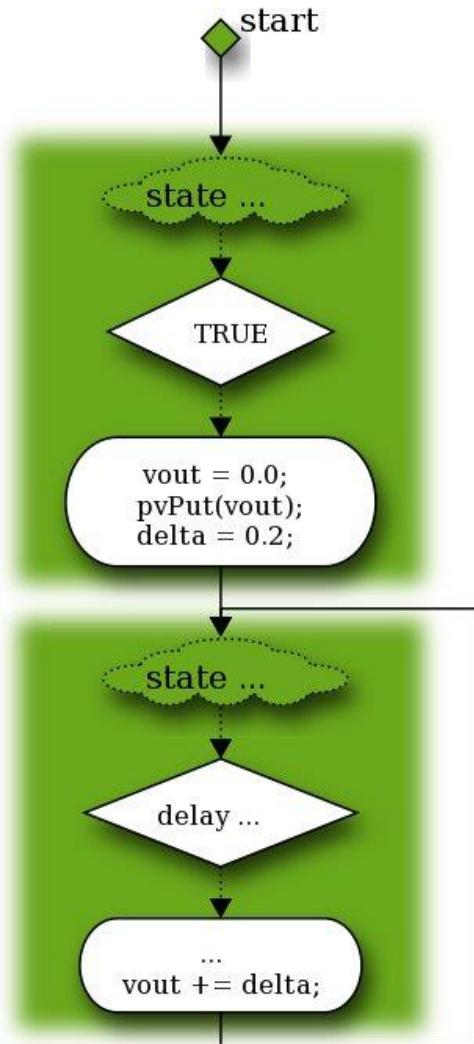
```
when (condition) {  
    /* transition actions */  
} state <next_state>
```

- C code may be ‘escaped’ (passed directly to the cc compiler)
 `%%` escapes a single line
 `%{ }%` escapes a block of code

SNL Basics

- Access to Process Variables via channel access is accomplished by assigning a variable to a PV
assign <var_name> to <process_variable_name>
- Variables used in when statements need their value to be automatically updated whenever a change occurs: this is declared using the monitor statement
monitor <var_name>
- PVs can be written to using pvPut
pvPut <var_name>

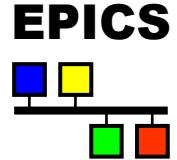
Additional State Set



Add a second state set to the previous example.

This state set generates a changing voltage value as its output: a triangle function with amplitude 11.

Complete State Program (with 2 state sets)



```
program level_check
```

```
float voltage;  
assign voltage to "ts1:ai1";  
monitor voltage;  
  
short light;  
assign light to "ts1:bo1";  
  
float vout;  
assign vout to "ts1:ao1";  
  
float delta;
```

```
ss volt_check {  
  
    state light_off {  
        when (voltage > 5.0) {  
            /* turn light on */  
            light = TRUE;  
            pvPut(light);  
        } state light_on  
    }  
  
    state light_on {  
        when (voltage < 3.0) {  
            /* turn light off */  
            light = FALSE;  
            pvPut(light);  
        } state light_off  
    }  
}
```

```
ss generate_voltage {  
  
    state init {  
        when () {  
            vout = 0.0;  
            pvPut(vout);  
            delta = 0.2;  
        } state ramp  
    }  
  
    state ramp {  
        when (delay(0.1)) {  
            if ((delta > 0.0 && vout >= 11.0) ||  
                (delta < 0.0 && vout <= -11.0) )  
                delta = -delta; /* change direction */  
            vout += delta;  
            pvPut(vout);  
        } state ramp;  
    }  
}
```

More Basics ...

- A state can have multiple when statements
- When entering a state, all when conditions are evaluated in the order given in the source code and the first one evaluated to be TRUE will be executed
- This allows conditional branching within the sequence program
- If no when condition is true, the sequence program task pends until an event occurs, which causes all when conditions to be re-evaluated
- It is easy to create a loop and consume all available CPU time
 - When this happens, all CA clients connected to this IOC will disconnect!

Example of “Multiple whens”

```

state checks
{
when(interlockChasPwrBI==0)
{
    sprintf(seqMsg1, "Electron Gun not ready ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "Gun Interlock Chassis off ");
    pvPut(seqMsg2);
} state initialChecks

when(gunLocal)
{
    sprintf(seqMsg1, "Electron Gun not ready ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2,"Egun in local control ");
    pvPut(seqMsg2);
} state initialChecks

when(gunInterlocksRdyCC==0)
{
    sprintf(seqMsg1, "Electron Gun not ready ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "Interlocks not OK ");
    pvPut(seqMsg2);
} state initialChecks

```

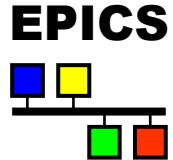
```

when(interlockChasPwrBI &&
      (gunLocal==0) && gunInterlocksRdyCC)
{
    gunAutoStart = 0;
    pvPut(gunAutoStart);
    gunAutoStop = 0;
    pvPut(gunAutoStop);
    sprintf(seqMsg1, "Push Auto-Start to begin autostart ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "Push Auto-Stop to begin autostop ...");
    pvPut(seqMsg2);
%% taskDelay(60);
} state waitForRequest
}

state initialChecks
{
when(delay(2.0))
{
    sprintf(seqMsg1, "Initial Checks");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "");
    pvPut(seqMsg2);
%% taskDelay(60);
} state checks
}

```

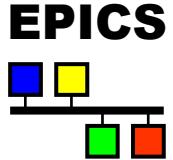
SNL Variables



Variables come in two variants:

- Local
 - Nested anywhere inside action blocks (state transition, entry, exit)
 - Temporary: re-initialized each time the block is entered
 - No assignment to PVs allowed
- Global
 - At the start of states and state set blocks
 - Local to the program, similar to “static” variables in C, but reentrant (if compiler option $+r$ is active)
 - *Can* be assigned & monitored

Other Features



- Assignment of macros at program startup for multiple copies of same sequence (must specify `+r` compiler flag)

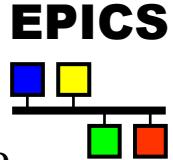
```
program level_check ("unit=ts1")  
  
float v;  
assign v to "{unit}:ai1";  
  
short light;  
assign light to "{unit}:bo1";
```

In startup script ...

```
seq &level_check, "unit=ts1"  
seq &level_check, "unit=ts2"
```

- Arrays (each element can be assigned to a PV)
- Built-In functions (see SNL Manual)
- Dynamic assignment of PVs to variables
- Connection Management and status

Events



An event is the condition on which statements following a **when** are executed and a state transition is made. Possible types of event:

1. **Change in value of a variable that is being monitored**

- Example: **when (achan < 10.0)**

2. **A timed event**

- Example: **when (delay(1.5))**

The delay value is in seconds, declared internally as a double and constant arguments **must** contain a decimal point.

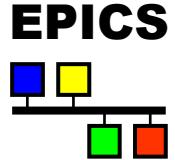
A delay is normally reset whenever the state containing it is exited.
Use the state specific **option -t** to keep it from being reset.

3. **Change in the *channel access connection status***

- Examples:

```
when(pvConnectCount() < pvChannelCount())
when(pvConnected(mychan))
```

Events (continued)



4. An internally generated event (*event flag*)

Examples :

```
when(efTestAndClear(myflag))  
when(efTest(myflag))
```

- Event flags can synchronize state sets within a program
 - Use **efSet(event_flag_name)** to set the flag
- They can also be associated with a monitored PV
 - Use **sync <monitored_variable> <event_flag>**
 - This allows an action to be handled on ANY change to the value
- Note that **efTest** does not clear the flag. **efClear** must be called sometime later to avoid an infinite loop.

Event Flags Example

```
double loLimit;
assign loLimit to "demo:loLimit";
monitor loLimit;
evflag loFlag;
sync loLimit loFlag;
```

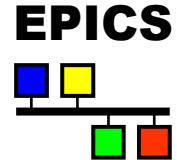
Create variable, assign to PV and monitor its value
Create event flag and sync to the variable

```
double hiLimit;
assign hiLimit to "demo:hiLimit";
monitor hiLimit;
evflag hiFlag;
sync hiLimit hiFlag;
```

Program ensures the condition
lowLimit <= highLimit is always met

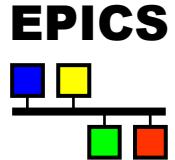
```
ss limit {
    state START {
        when ( efTestAndClear( loFlag ) && loLimit > hiLimit ) {
            hiLimit = loLimit; pvPut( hiLimit );
        } state START
        when ( efTestAndClear( hiFlag ) && hiLimit < loLimit ) {
            loLimit = hiLimit; pvPut( loLimit );
        } state START }
}
```

Additional Features



- Macros:
 - assign Vout to "{unit}:OutputV";
(must use the +r compiler options for this if more than one copy of the sequence is running on the same IOC)
ts1> seq &example, "unit=HV01"
- Some of the SNC compiler options:
 - +r make program reentrant (default is -r)
 - -c don't wait for all channel connections (default is +c)
 - +a asynchronous pvGet() (default is -a)
 - -w don't print compiler warnings (default is +w)
 - +e eftest automatically clears flag (default is -e)
 - +d turn on run-time debug messages

Additional Features (continued)



- Pass parameters to programs at run time:

```
ts1> seq &example, "bias = 2.55"
```

```
pStr = macValueGet("bias"); /* this SNL statement gets passed in value */
```

- Access to alarm status and severity:

- `pvStatus(var_name)`
- `pvSeverity(var_name)`

- Queuable monitors (V2 feature):

- Saves monitors in a queue in the order they come in, necessary if you care about fast transitions
- `syncQ <var_name> <event_flag>` [optional length of the queue]
- `pvGetQ(var_name)`
Removes oldest value from variable's monitor queue
- `pvFlushQ(var_name)`
Clears the queue

Debugging

seqShow

```
ioc> seqShow
```

Program Name	Thread ID	Thread Name	SS Name
xx_RF_Cond	10854616	xx_RF_Cond	11AutoConditioning
bpmTraject	10838832	bpmTrajectory	bpmTrajectorySS
xx_autoPha	10680172 10573424	xx_autoPhasing xx_autoPha_1	autoPhasing updatePresets
xx_autoRfT	10589876	xx_autoRfTiming	autoRfTiming
value = 0 = 0x0			

Debugging

seqShow <taskID>

```
ioc> seqShow 10838832
State Program: "bpmTraject"
initial task id=10838832=0xa56330
thread priority=100
number of state sets=1
number of channels=56
number of channels assigned=56
number of channels connected=56
options: async=0, debug=0, newef=0, reent=0, conn=0
log file fd=3
log file name="/tyCo/0"

State Set: "bpmTrajectorySS"
task name=bpmTrajectory;  task id=10838832=0xa56330
First state = "init"
Current state = "waitToPlot"
Previous state = "plotWithBeam"
Elapsed time since state was entered = 0.4 seconds)

value = 0 = 0x0
```

Debugging

seqChanShow <taskId>

```
ioc> seqChanShow 10838832
State Program: "bpmTraject"
Number of channels=56

#1 of 56:
Channel name: "L1:PG1:PM1:BPM.XPOS"
    Unexpanded (assigned) name: "L1:PG1:PM1:BPM.XPOS"
    Variable name: "L1PG1PM1_X"
        address = 11931404 = 0xb60f0c
        type = float
        count = 1
        Value = 0
        Monitor flag=1
            Monitored
            Assigned
            Connected
            Get not completed or no get issued
            Status=11
            Severity=2
        Time stamp = 05/21/99 16:43:35.085407596
        Next? (+/- skip count)
```

Summarize CA Status

```
ioc> seqcar
```

```
Total programs=1, channels=1, connected=0,  
disconnected=1
```

- This produces a report about all the channel access connections from sequence programs

Debugging

- Print messages to the IOC console
 - `printf("Here I am in state xyz \n");`
- Report debugging messages to String PVs:
 - `sprintf(seqMsg1, "Here I am in state xyz");`
 - `pvPut(seqMsg1);`
- Stop and restart
 - `seqShow`
 - `seqStop <taskID>`
 - `seq &my_sequence_program`

Building a state program

1. Use editor to create the source file in “*exerciseApp/src*”
 - ❖ File name must end with “.stt”, e.g. “*exercise9.stt*”.

2. Examine “*exerciseApp/src/Makefile*

...

```
#exercise_DBD += exercise9.dbd
```

```
# exercise_registerRecordDeviceDriver.cpp will be created
exercise_SRCS += exercise_registerRecordDeviceDriver.cpp
```

```
exercise_SRCS += exerciseMain.cpp
```

```
# Add locally compiled object code
```

```
exercise_SRCS += exercise8.c
```

```
#exercise_SRCS += exercise9.stt
```

...

Uncomment the lines which contain “*exercise9*”.

Type “*make*”. This runs the State Notation Compiler and builds the code into the IOC binary.

The Run-Time Sequencer

1. The sequencer executes the state program in the EPICS IOC environment.
2. The sequencer supports the event-driven execution; no polling needed.
3. Each state set becomes an OS task (or thread).
4. The sequencer manages connections to database channels through Channel Access.
5. The sequencer provides support for Channel Access (put, get, and monitor).
6. The sequencer supports asynchronous execution of delay and event flag functions.
7. Only one copy (object module) of the sequencer is required on each IOC.
8. Query commands display information about executing state programs.

EPICS **Channel Access** **Gateway**

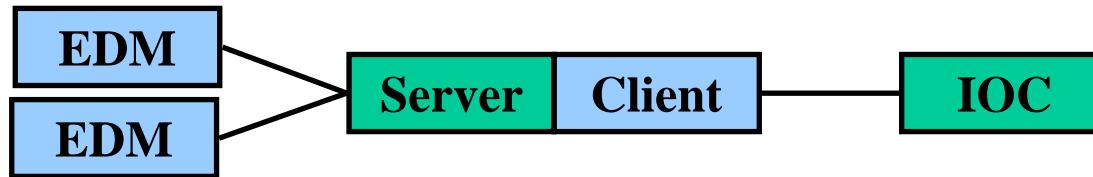
Philip Taylor

Observatory Sciences Ltd

*Based on original material by Ken Evans & Marty
Smith (APS) and Pete Leicester (DLS)*

What is the Gateway ?

- **Both a Channel Access server and a Channel Access client**
 - Clients such as EDM connect to the server side
 - Client side connects to remote servers such as IOCs

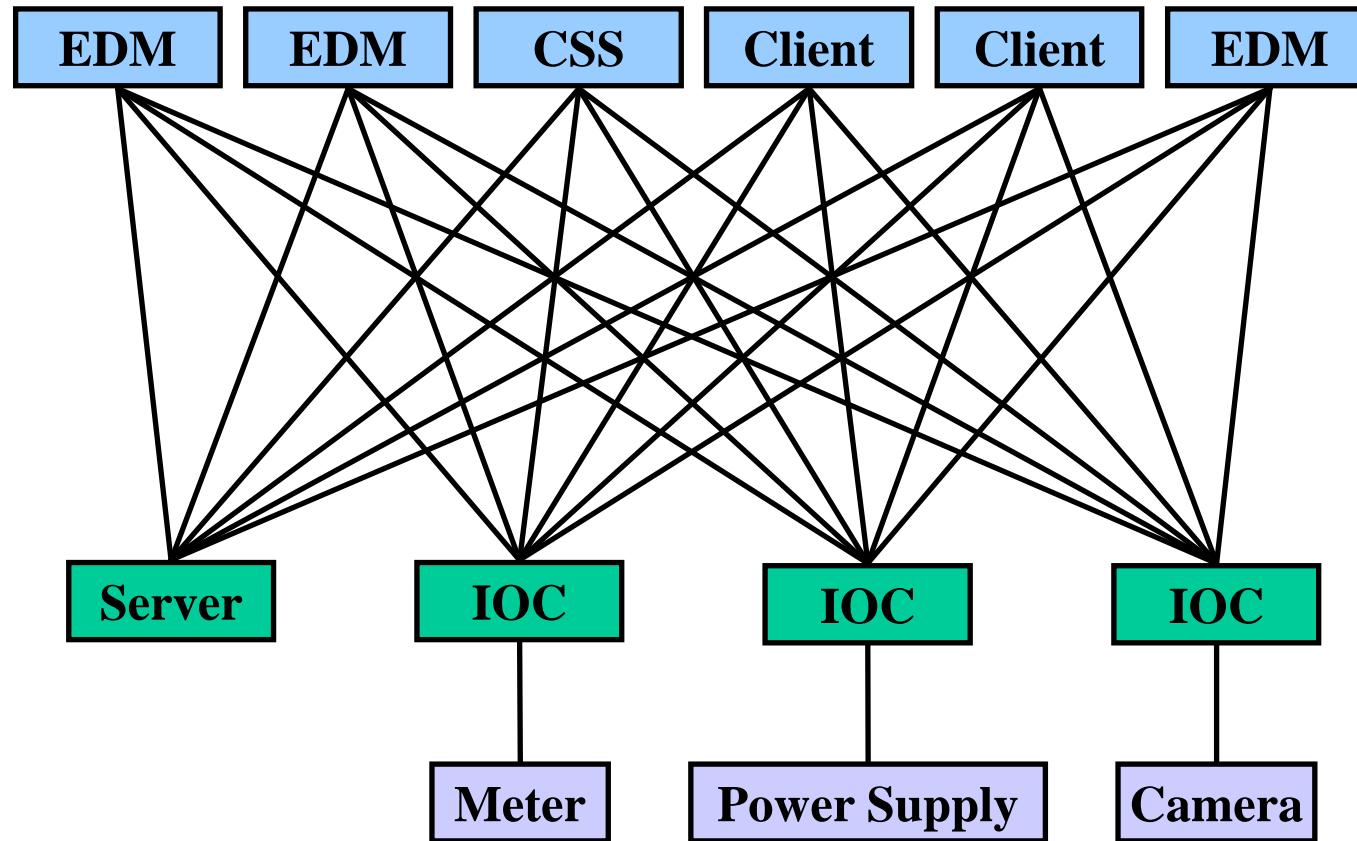


- **Only one CA connection between gateway and IOC rather than one per client**
 - Minimizes TCP connections to IOCs
 - vxWorks has open file limit
- **Read access from clients is answered from data cache**
 - Reduces network traffic for read requests
 - IOC sends monitor events only to the gateway
- **CA connection is held open by the gateway after last client disconnects**
 - Time is configurable

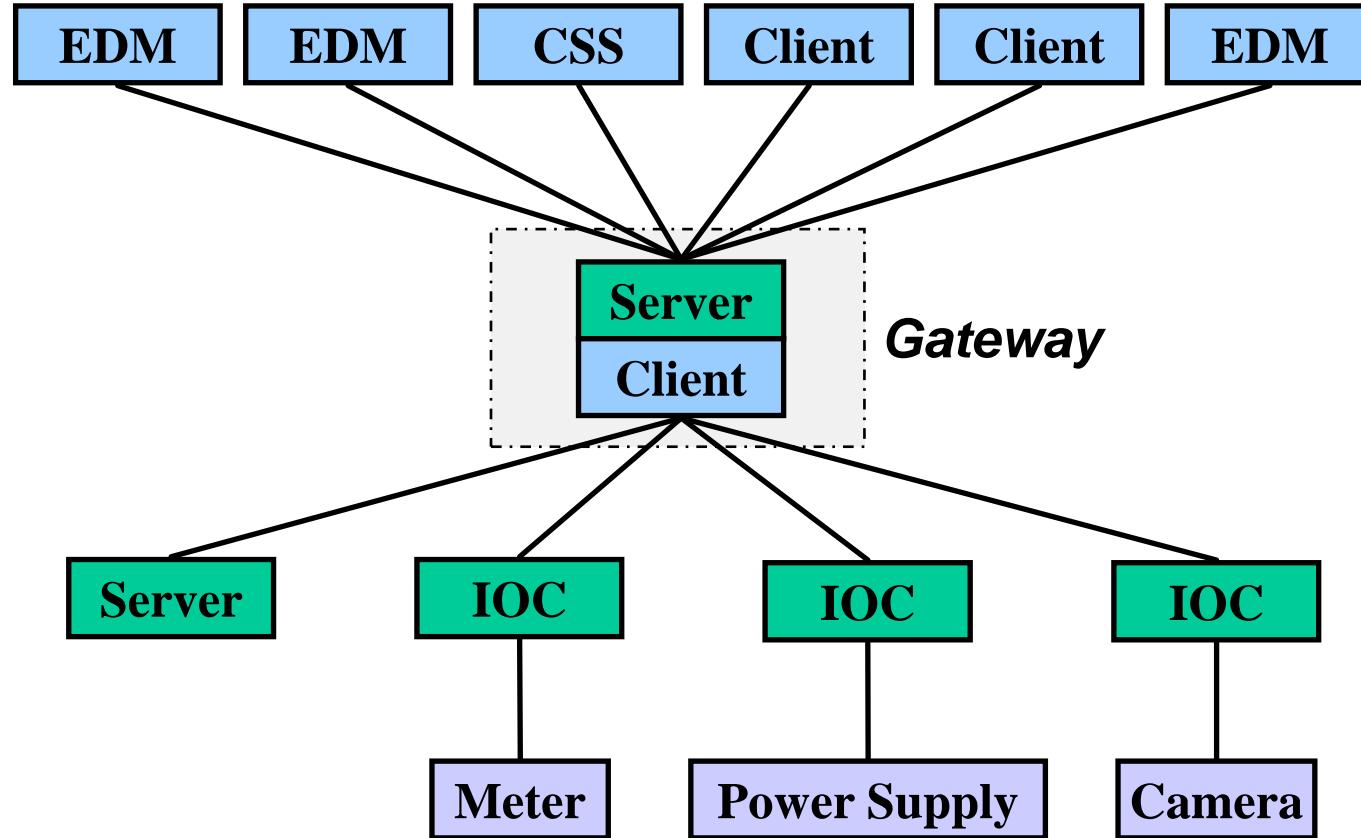
What is a PV Gateway Good For?

- **Getting CA clients on one network to connect to CA servers on another network**
 - Typically the gateway host computer has multiple Ethernet interfaces on different subnets
- **Aliasing PV names from the real PV name**
- **Access Security**
 - Supplements IOC level access security
 - Restrict access to certain PVs from certain hosts
 - Restrict access to read only from certain networks
- **CA ‘put’ logging**
 - Can see who changes a PV and from where change was made

EPICS Overview



Gateway



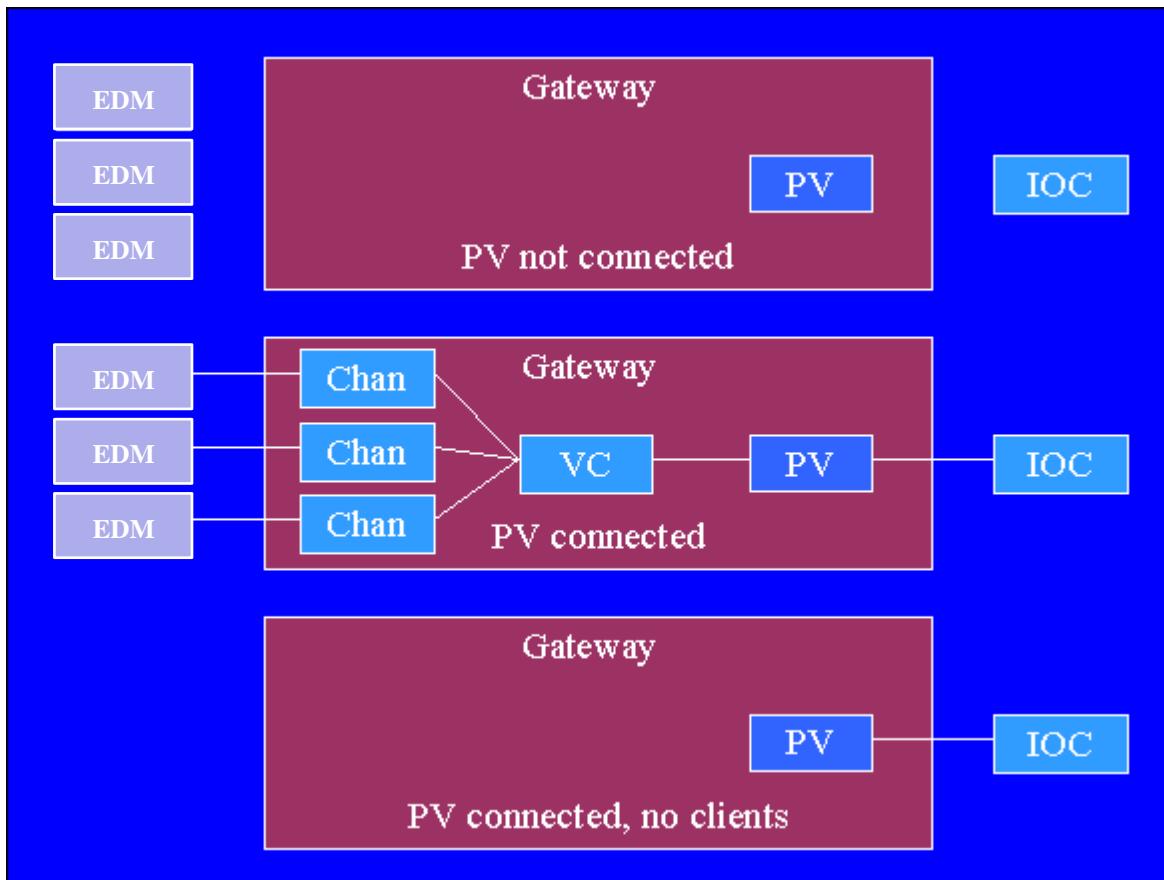
Gateway Internals : States and Objects

- **PV (Process Variable)**

Objects handle Gateway client connections to IOC

- **VC (Virtual Connection)**

Objects handle Gateway server connections to clients



1. No connection to IOC (lost/connecting)
2. Typical – IOC and client connections established
3. PV connected to IOC, but no clients interested
(*PV object & connection maintained for 2 hours*)

Gateway Access Security

- The Gateway can restrict access to PVs from specific hosts
- It supplements but cannot override IOC access security
- The configuration is specified in three files that are read at startup:
 1. *gateway.command*
Specifies commands to execute when gateway receives the USR1 signal (i.e. kill –USR1 <pid>)
 2. *gateway.access*
Same format as IOC access security (see the AppDev guide). Defines access security groups and levels
 3. *gateway.pvlist*
Specifies which PV name patterns are allowed or denied using regular expressions

gateway.pvlist Format

Lines in this file have one of the following formats:

1 . EVALUATION ORDER <eval-order>

- Sets precedence when matching ALLOW and DENY rules both found

2 . <pv-name-pattern> DENY [FROM] [<host> ...]

- Ignores requests for PVs matching the pattern
- ‘DENY FROM’ blocks requests from specific hosts

3 . <pv-name-pattern> ALIAS <real-pv-name> [<asg> [<asl>]]

- Creates and serves a PV alias with optional access security

4 . <pv-name-pattern> ALLOW [<asg> [<asl>]]

- Declares PVs the gateway should serve, with optional access security

DLS gateway.pvlist file

Example: access to Beamline PVs from Primary (machine) network

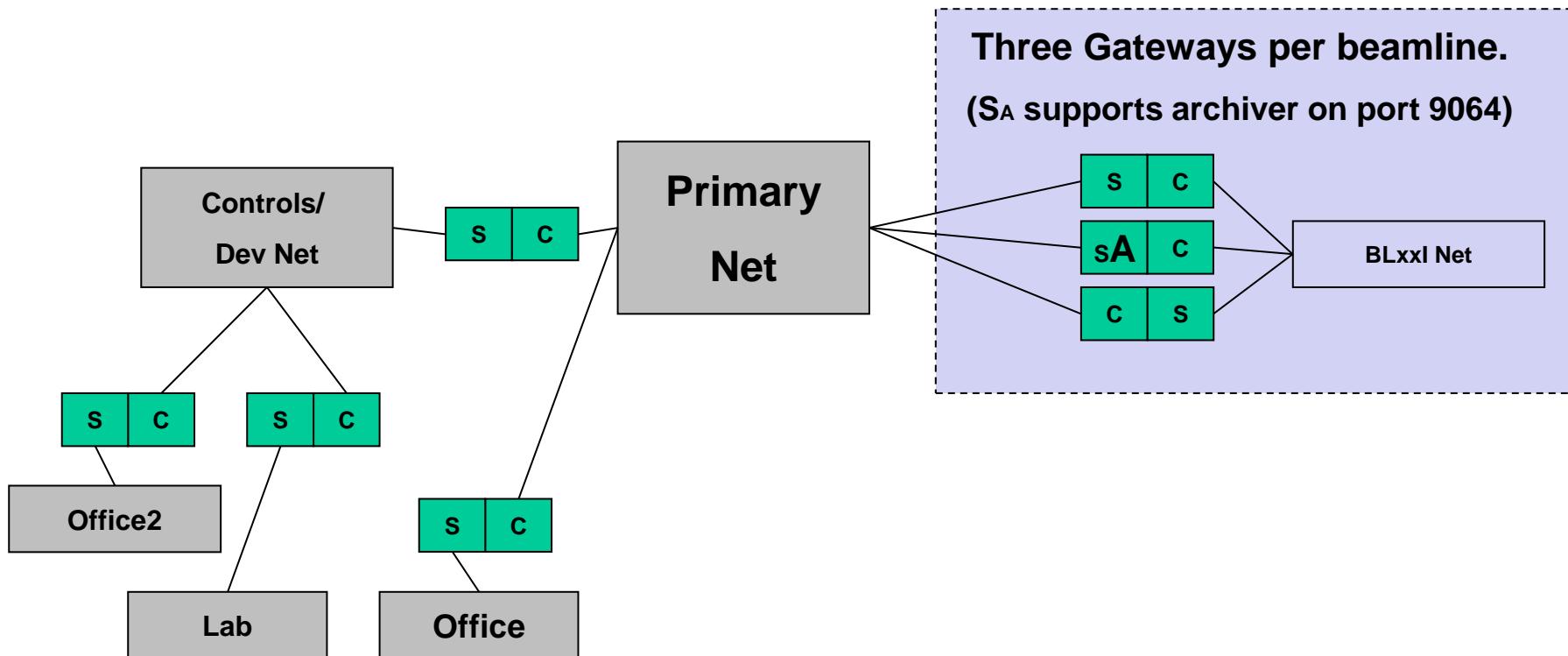
- # Deny takes precedence
- EVALUATION ORDER DENY, ALLOW

- #BL[0-2][0-9][IJB]-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- BL22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- BL22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- BL22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- BL22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- BL22I-[A-Z][A-Z]-[A-Z0-9]-[0-9][0-9].* ALLOW

- # Beamline IOC's have some PV's beginning SRxxI (from BLInterface module)
- SR22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- SR22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- SR22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- SR22I-[A-Z][A-Z]-[A-Z0-9]-[0-9][0-9].* ALLOW
- SR22I-[A-Z]-[A-Z0-9]-[0-9][0-9].* ALLOW

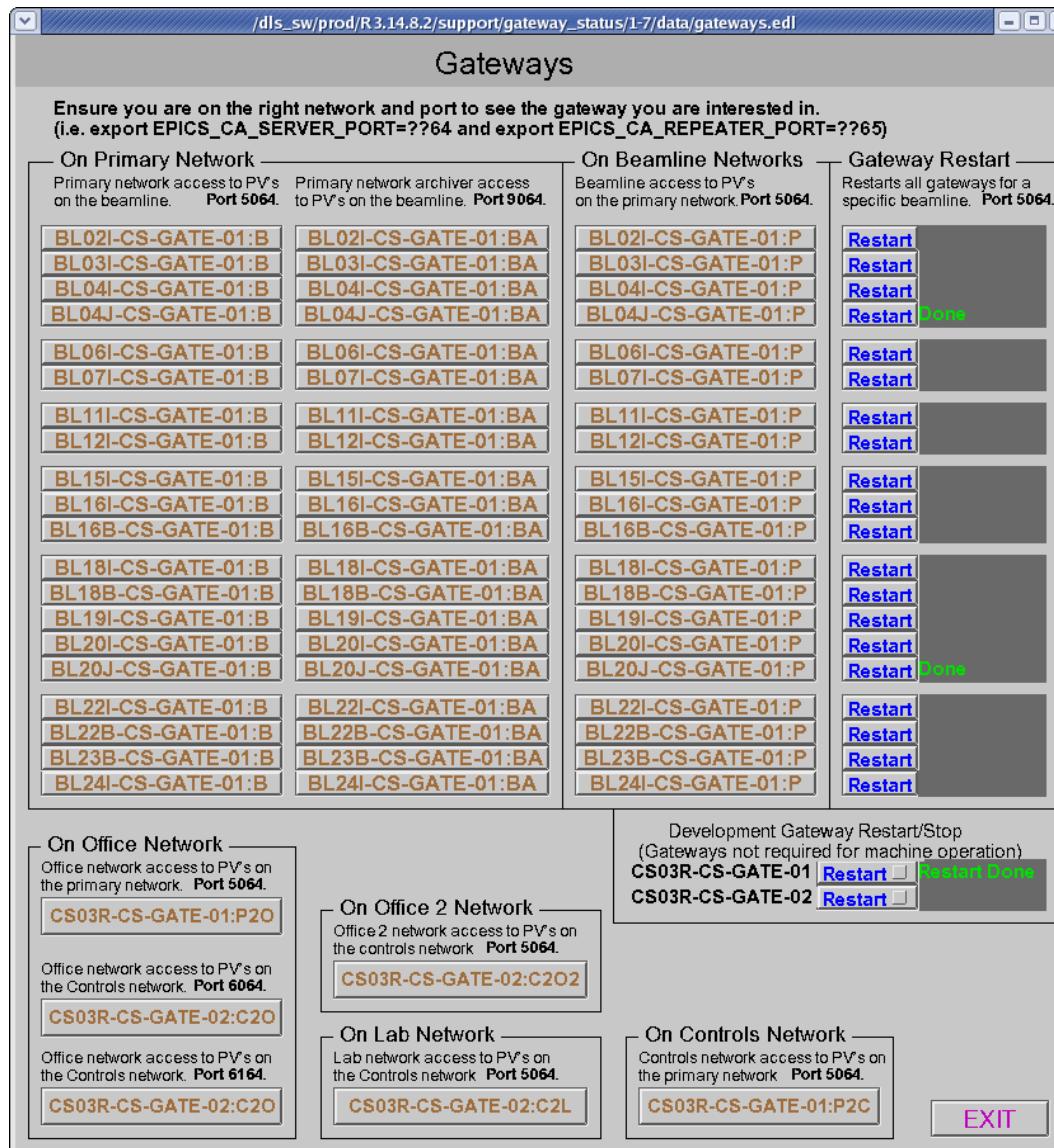
- # Beamline IOC's have some PV's beginning FExxI (from BLInterface module)
- FE22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- FE22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- FE22I-[A-Z][A-Z]-[A-Z0-9][A-Z0-9][A-Z0-9]-[0-9][0-9].* ALLOW
- FE22I-[A-Z][A-Z]-[A-Z0-9]-[0-9][0-9].* ALLOW
- FE22I-[A-Z]-[A-Z0-9]-[0-9][0-9].* ALLOW

DLS Gateway Architecture

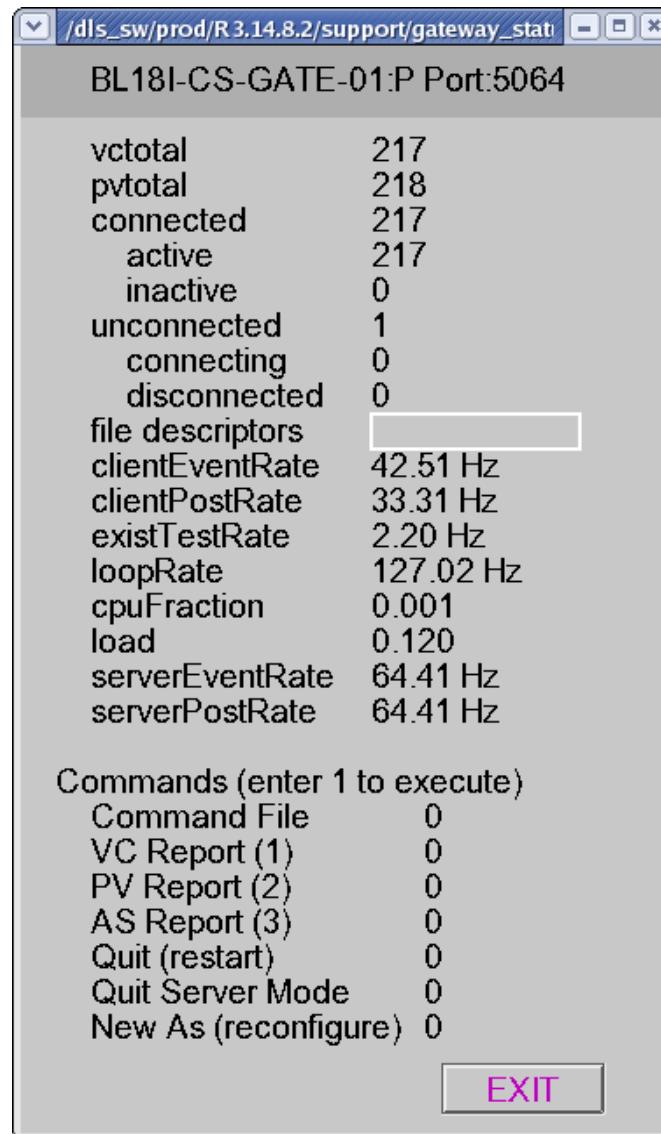


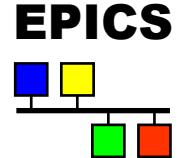
DLS Gateway Status

(Launcher: Utilities -> Gateway Status)



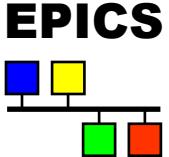
Gateway Status





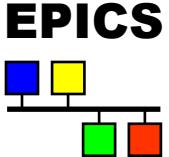
EPICS Motor Support

Andy Foster
Observatory Sciences Limited



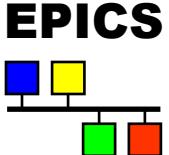
EPICS Motor Support

- Interface to the user is the EPICS **motor record**
 - Lots of code has been written to interface to this object
 - Java, IDL, Python classes, etc.
- Next layer is EPICS **device support**
 - Knows about the motor record, talks to the driver
- Lowest layer is EPICS **driver**
 - Knows nothing about motor record, but talks to the motion controller
- 3 models of device and driver support have developed over time



Model 1 (pre-Asyn)

- Device support and driver support which were controller specific (OMS – Oregon Micro Systems now OMS Motion)
- Interface between device and driver support is custom for the controller
- But cannot take advantage of controller-specific features not supported by motor record
- No provision for multi-axis coordination



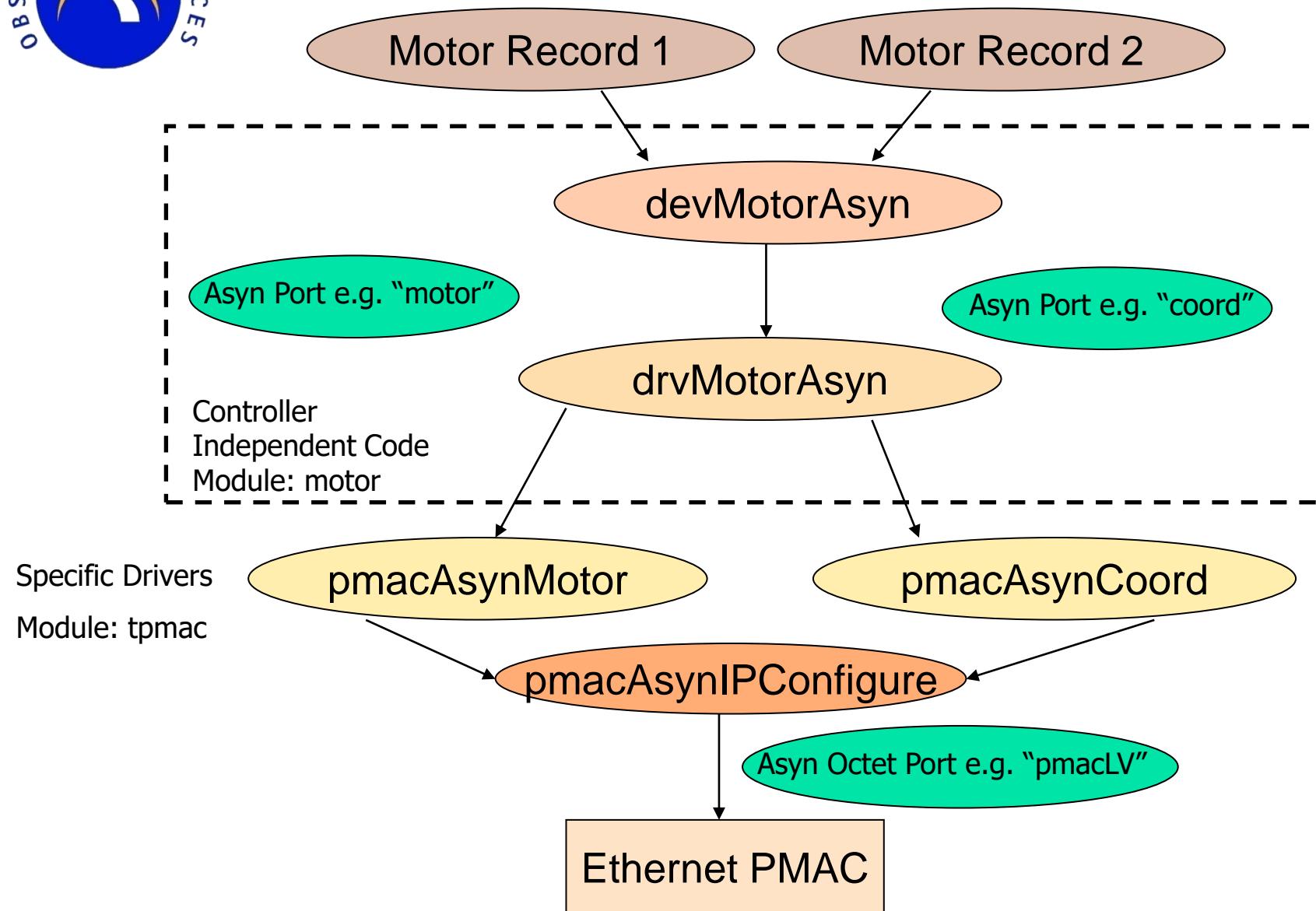
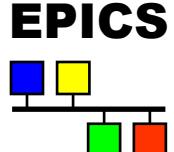
Model 2 (post-Asyn)

- Developed 2006 by APS and Diamond
- Uses standard *asyn* interfaces to communicate between device support and driver
- A *single device-independent* device support file
- A *single device-independent* driver support file for asyn interfaces
- Below device-independent driver is a device specific one i.e. a driver for OMS, a driver for PMAC etc
- No implementation of multi-axis coordination
- Many drivers in the motor package have a Model 2 implementation

Newport XPS, OMS, Attocube ANC150, Aerotech Ensemble, **PMAC**



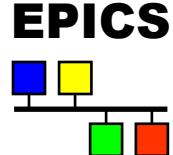
Model 2 motion control: software layers





Example Model 2

Startup Script



```
# Send Octet strings to the PMAC controller through this asyn port
# (asyn octet port, "ethernet address:port")
pmacAsynIPConfigure("pmacLV", "172.23.243.143:1025")

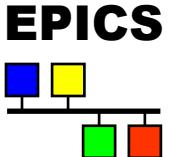
# Set up PMAC motor controller on asyn octet port
# (asyn octet port, asyn address, controller id, numAxes)
pmacAsynMotorCreate( "pmacLV", 0, 1, 8)

# Independent motor driver code. Calls into PMAC specific motor driver functions
# "pmacAsynMotor", using "motor" asyn port.
# (asyn port, driver name, controller id, numAxes+1)
drvAsynMotorConfigure("motor", "pmacAsynMotor", 1, 9)

# Set up PMAC Coordinate system controller on asyn octet port
# (asyn octet port, asyn address, CS Num, ref, motion program number)
pmacAsynCoordCreate( "pmacLV", 0, 5, 11, 10 )

# Independent motor driver code. Calls into PMAC specific coordinate system motor driver
# functions "pmacAsynCoord", using "coord" asyn port.
# (asyn port, driver name, ref, numAxes+1)
drvAsynMotorConfigure("coord", "pmacAsynCoord", 11, 9)

iocInit()
```



EPICS Motor Record

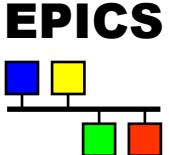


General user interface to most kinds of motor:

- Move, Stop, Home
- User offset
- Change sign of motion
- Motion retries in case of failed move
- Handles backlash
- Software limits
- Axis status
- Set velocity, accel, etc

All logic is in:
motor/motorApp/MotorSrc/motorRecord.cc

>100 fields.



Device – Independent layer

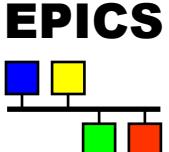
`devMotorAsyn.c` - motor module (motor/motorApp/MotorSrc)

At iocInit (when EPICS starts):

- Initialise the motor record with information from driver
- Connect to asyn port, get handles to asyn interfaces (eg. asynInt32) created by the driver layer.

Running:

- Update motor record when callbacks arrive from the driver (position, velocity, axis status bits).
- Format commands from motor record and writes them into the asyn port buffer for the driver.



Driver: Independent Layer

drvMotorAsyn.c - motor module (motor/motorApp/MotorSrc)

Implements asyn interfaces e.g. writeInt32

Create asyn port to communicate with device support

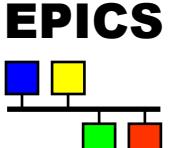
Gets handle, "drvset", for specific driver e.g. pmacAsynMotor, pmacAsynCoord

Picks out driver function depending on pasynUser->reason

```
static asynStatus writeInt32(void *drvPvt, asynUser*pasynUser, epicsInt32 value)
{
    drvmotorPvt      *pPvt = (drvmotorPvt *)drvPvt;
    drvmotorAxisPvt *pAxis;

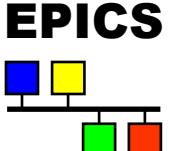
    pasynManager->getAddr(pasynUser, &axisNumber); // device address this asynUser talks too
    pAxis          = &pPvt->axisData[axisNumber];
    command        = pasynUser->reason;           // Index into all possible commands

    switch(command)
    {
        case motorStop:
            (*pPvt->drvset->stop)(pAxis->axis, pAxis->accel);
            break;
    }
}
```



Driver: Controller Specific Layers

- **pmacAsynMotor and pmacAsynCoord**
 - Each implements a set of function pointers to perform certain actions e.g. stop/home/move.
 - Essentially, a driver entry table for PMAC
 - Starts thread to poll controller for axis information (position, velocity, status)
 - MovingPollPeriod default 10Hz
 - IdlePollPeriod default 2Hz

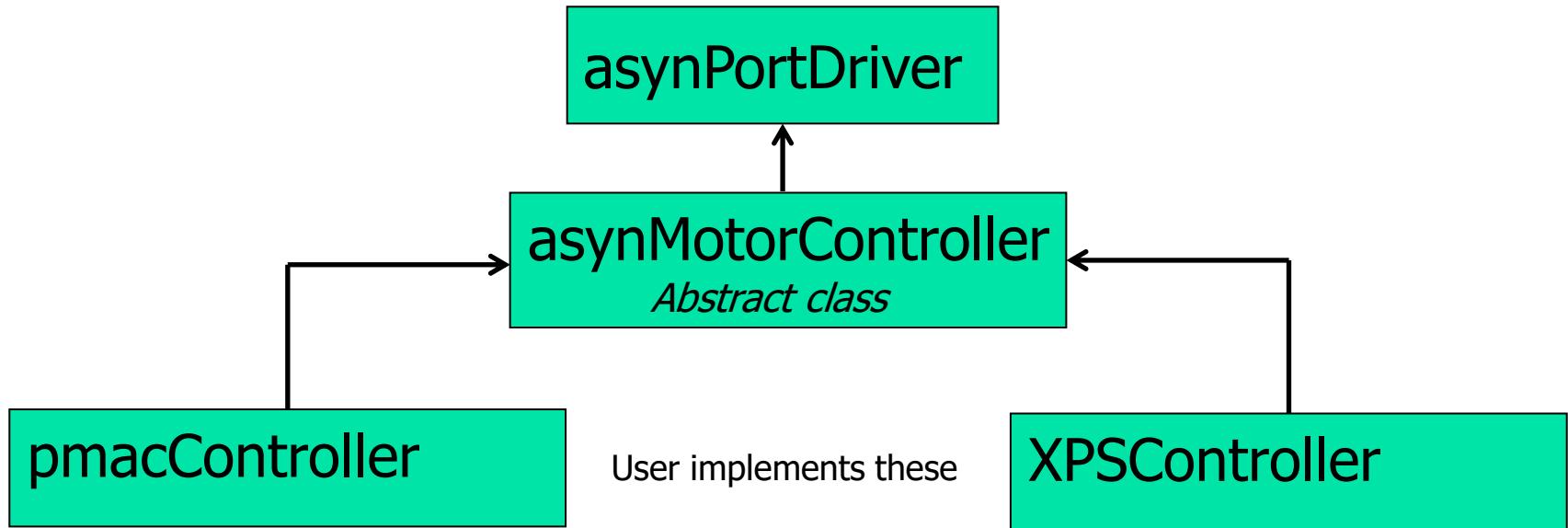
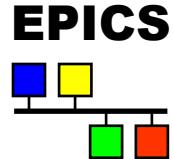


Model 3 (AsynPortDriver)

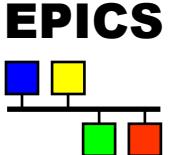
- New C++ model.
- Two base classes
 - asynMotorController (inherits from asynPortDriver)
 - asynMotorAxis (new) takes asynMotorController object and axis number as arguments.
- Base classes provide much functionality, only need to write device-specific implementations.
- Easy to support controller-specific features
- Direct support for coordinated profile moves in the driver API.



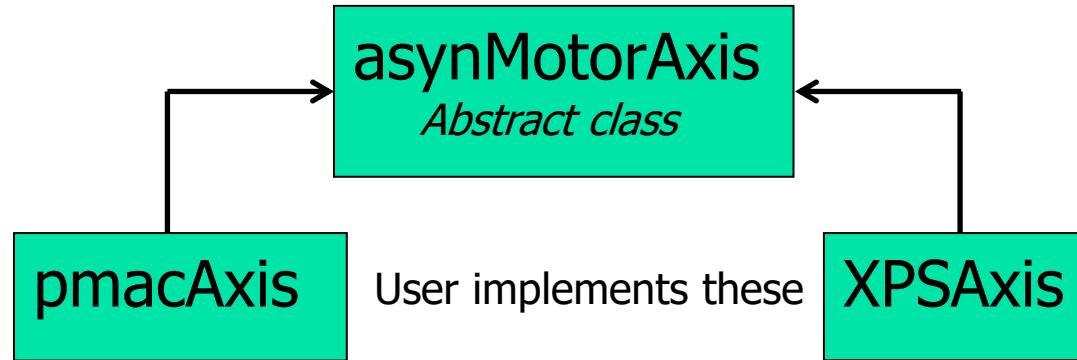
Model 3 Controller Class Hierarchy



- Use `asynOctetSyncIO writeRead` interface for communication to the controller.
- Logic that is controller wide (eg. deferred moves involves more than 1 axis)
- Provides default parameters (velocity, acceleration, soft limits, etc).
- Task to poll for controller status



Model 3 Axis Class Hierarchy

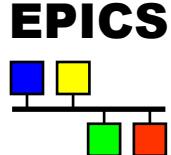


Implements a set of functions to cause actions to the axes:

- Move
- Stop
- Home
- ...
- Task to poll for axis status

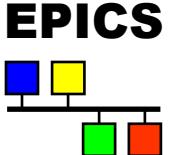


New Model 3 PMAC driver



Actually has 5 important classes:

- pmacController - controller class
- pmacAxis - axis class
- pmacCSController – coordinate system controller class
- pmacCSAxis – coordinate system axis class
- messageBroker
 - handles all communication with PMAC, above classes register with this
 - Reduces number of threads in driver
 - Provides better control of messages sent to the PMAC
 - Simplifies logic

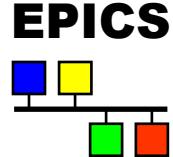


New Model 3 PMAC driver

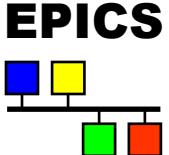
- This driver is the support module called “pmac”
 - As opposed to “tpmac” which is the old model 2 driver
 - Can do trajectory scanning, coordinate systems all in one driver
- Can use standard EPICS records to control velocity, move the motor...
 - Don’t have to just use the motor record (because of parameter library)
 - But motor record maintains a lot of state information e.g. backlash etc
- Currently being used on beamlines which are involved in Hardware Triggered Scanning (Mapping)
 - We should be using this everywhere now rather than “tpmac”.



New Model 3 PMAC Driver features



- Coordinate system support
 - CS Definitions, axis assignments
 - Kinematics (how we do virtual axes)
 - Save/Load CS configurations
- Slow, Medium and Fast polling loops
 - Place PMAC variables on these loops to watch for changes
- Extensive debugging information
- Deferred move support for CS motors as well as real motors
- STOP – stops CS motors as well as real motors (previously not supported)!



Model 3 - Example IOC startup

- #Low level IP port
- pmacAsynIPConfigure("geo1", "172.23.88.164:1025")

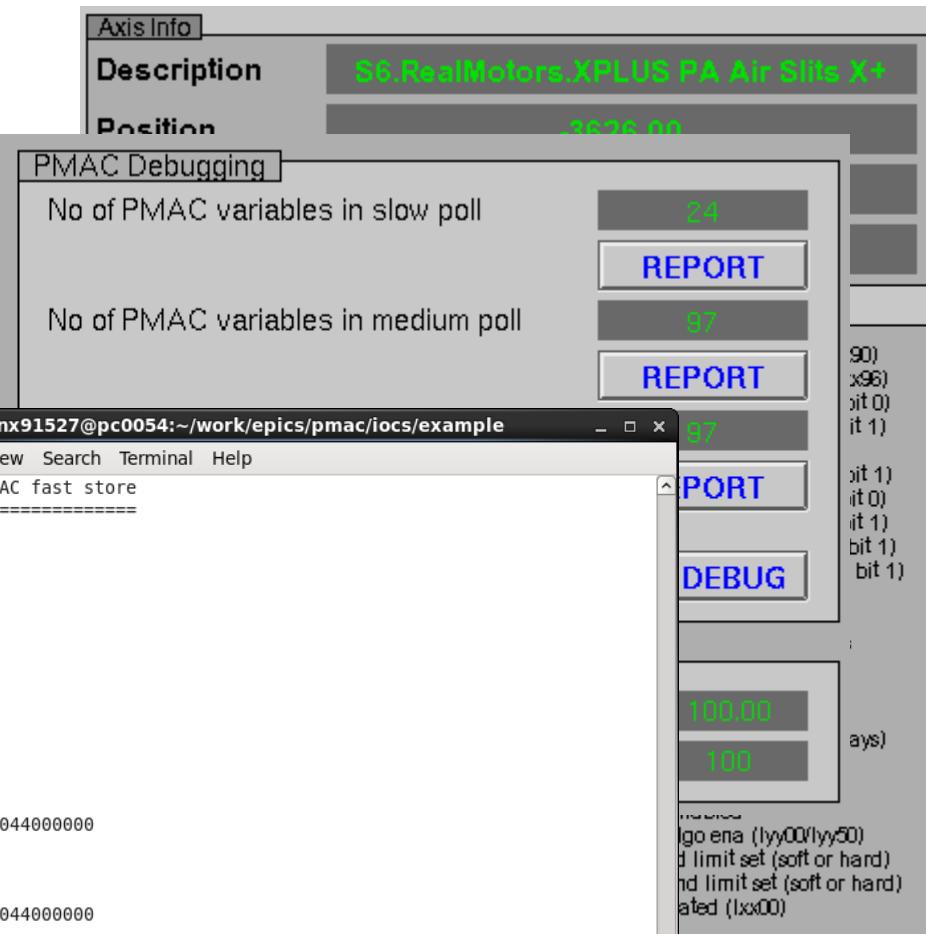
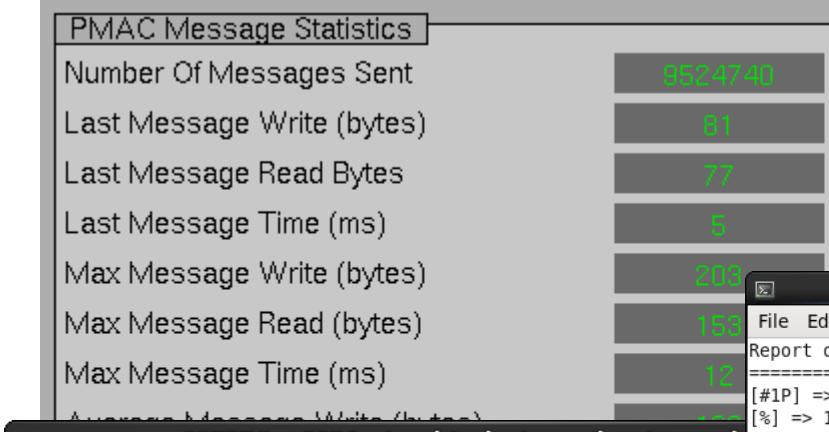
- #PMACCreateController(motor record port name, low level port, low level port address, num axes, moving polling period (ms), idle polling period (ms))
- pmacCreateController("PMAC1", "geo1", 0, 8, 50, 500)

- #pmacCreateAxis(motor record port name, axis number)
- pmacCreateAxis("PMAC1", 1)
- pmacCreateAxis("PMAC1", 2)
- #Etc... just poll for number of axes being used!
- #Or, can use:
- #pmacCreateAxes(motor record port name, max number of axes).
- pmacCreateAxes("PMAC1", 8)



Screenshots

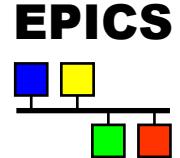
EPICS



```
gnx91527@pc0054:~/work/epics/pmac/iocs/example
File Edit View Search Terminal Help
Report of PMAC slow store
=====
[P4009] => 0
[P4008] => 0
[P468] => 0
[P469] => 0
[I58] => 0
[P465] => 0
[I5] => 2
[I56] => 0
[P466] => 0
[P467] => 0
[P4004] => 0
[I3] => 2
[I6] => 1
[P471] => 0
[P470] => 0
[P472] => 0
[I90] => $39
[I95] => $7
[I97] => $0
epics> █
```

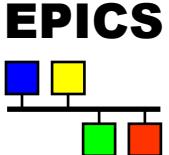
```
gnx91527@pc0054:~/work/e  
File Edit View Search Terminal Help  
Report of PMAC fast store  
=====
```

[#1P] => 0
[%] => 100
[&3086] => 0
[&1Q89] => 0
[&1Q88] => 0
[&1Q87] => 0
[&1Q86] => 0
[#3P] => 0
[&1Q84] => 0
[#5P] => 0
[&1Q82] => 0
[&1Q81] => 0
[P1001] => 0
[#8?] => 840044000000
[#1F] => 0
[#5F] => 0
[#3F] => 0
[#6?] => 840044000000
[#1V] => 0
[#4?] => 840044000000
[i124] => \$820401
[P4001] => 0
[#2?] => 840044078000
[&1Q85] => 0
[P4005] => 0
[P4007] => 0
[P4006] => 0
[#7V] => 0
[i1324] => \$820401
[&3??] => A80020000000000000000
[&3Q88] => 0
[&1Q83] => 0



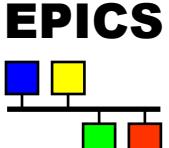
Motor Record Configuration

Andy Foster
Observatory Sciences Limited



The EPICS Motor record

- The EPICS **motor record** is rather large
- Latest version, 7-2-1, contains 4125 lines of code!
- Goes against the ethos of EPICS records, small function blocks, easy to understand
- goto statements – not easy to follow the logic
- Has developed “piecemeal”

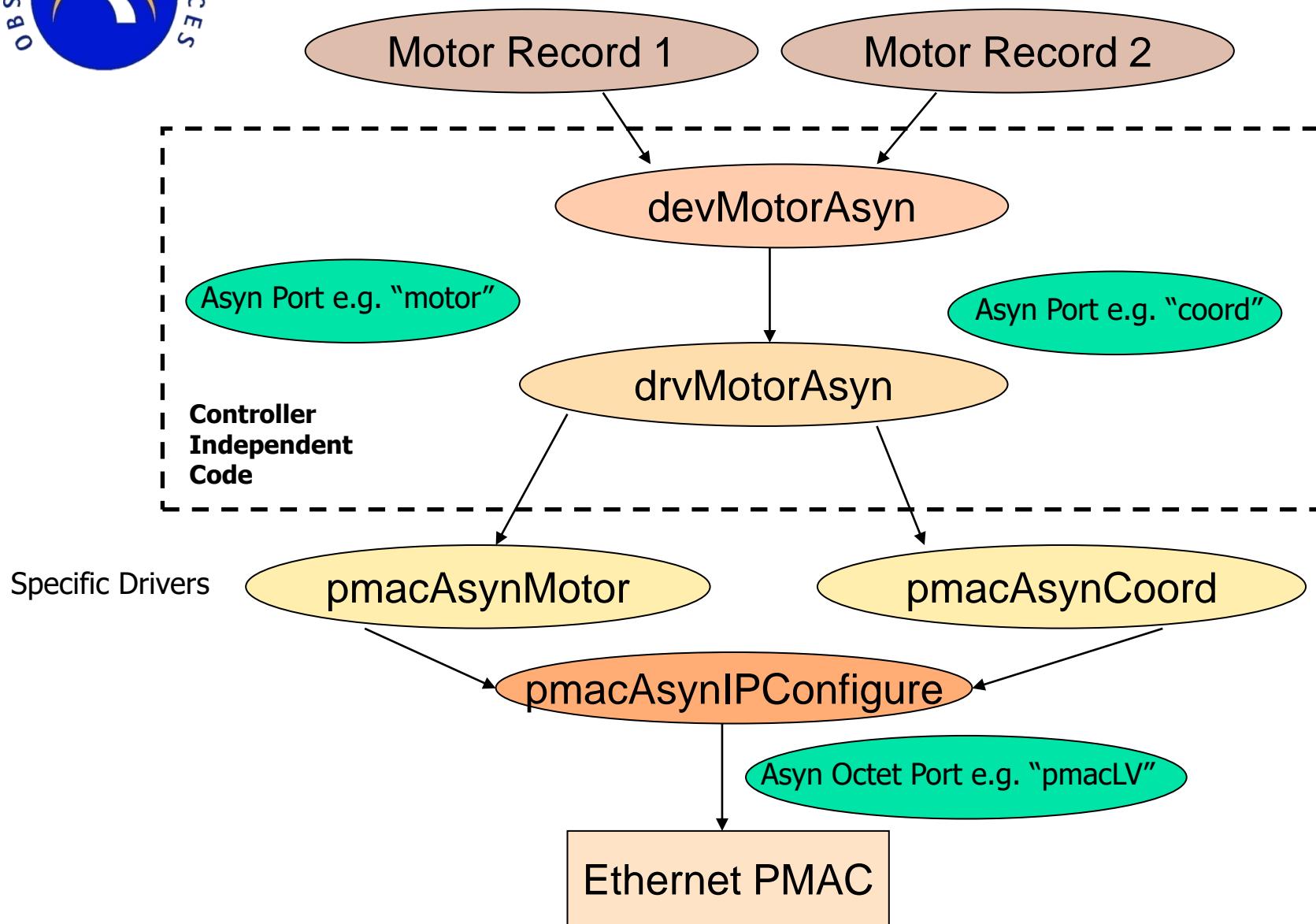
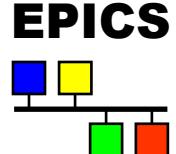


The EPICS Motor record

- Developed originally for the OMS motion controller
- Adding new functionality very difficult, easy to break something
- Original developer: (early 90's) : Jim Kowalkowski (APS)
Tim Mooney (APS)
- Now on github: <https://github.com/epics-modules/motor/>
- There are 121 fields in the record, which ones are important?
- On the positive side, it provides a uniform interface to a motor, very important to have this in a large facility

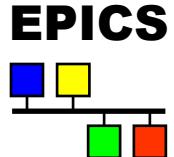


Model 2 motion control: software layers

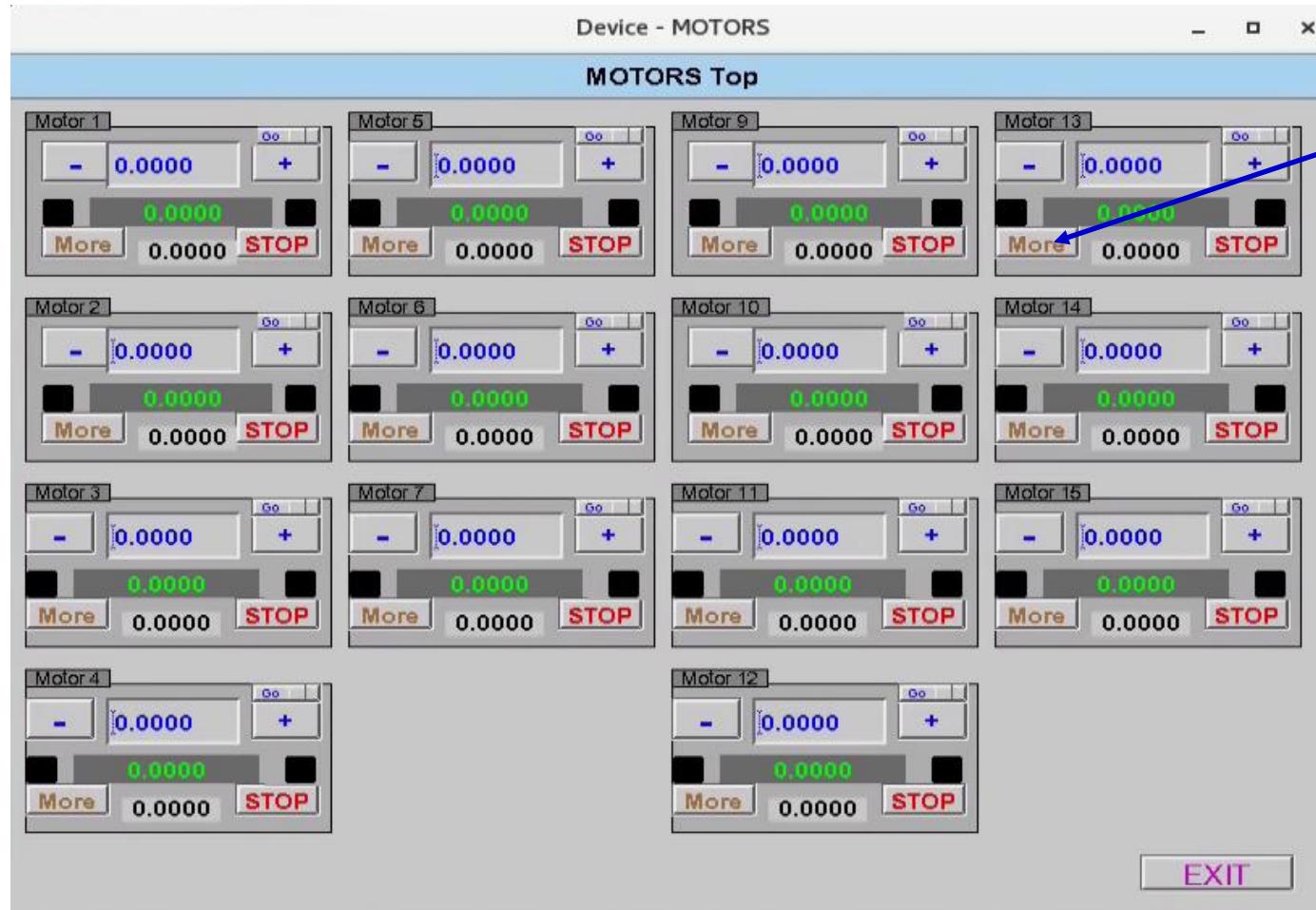




Interactive Motor Record GUI



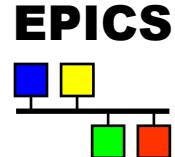
/dls_sw/work/R3.14.12.7/support/ajf67/EPICSEExercises/startMotionSimGUI



Click on
“More”



EPICS Motor Record GUI



Demand
VAL



Readback
RBV

Status bits
MSTA



Motor Record Controls

- Stop**

closes the loop on the controller ("#1j/")

- Home Reverse and Forward**

Both do the same thing ("#1hm").

Direction decided by Ixx23 on PMAC

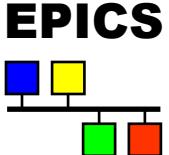
- Jog Reverse and Forward**

Sends "#1j-" and "#1j+" for as long as they are pressed

- Tweak Reverse and Forward**

Adds or subtracts the Tweak Step to the current demand and sends it





Motor Record Controls

- **Kill**

Sends a kill to the controller ("#1k")

- **Sync VAL=RBV**

Set the VAL field equal to the Readback.
Can be out of step after homing.

- **Pos/Neg**

Switches positive and negative directions.

Changes the offset in EPICS if Offset=Variable. Changes the sign of the current position in EPICS if Offset=Frozen.

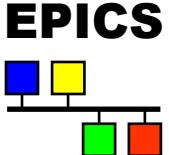
Confusing – **Do not** change!

- **Use/Set**

Determines where the demand is actually written too. "Set" mode changes Offset in EPICS if Offset=Variable. Changes hardware position on PMAC if Offset=Frozen! Confusing – **Leave as Use!**



These open further screens



Motor Record Controls

Offset	<input checked="" type="checkbox"/> Variable	Use Encoder	<input type="checkbox"/> No
--------	--	-------------	-----------------------------

- **Offset (Variable/Frozen)**

Motor record maintains two coordinate systems: User and Dial.

$$\text{UserVAL} = \text{DialVAL} * \text{DIR} + \text{OFFset}$$

Frozen prevents **Pos/Neg** and **Use/Set** from changing the offset. Normally leave this as Variable.

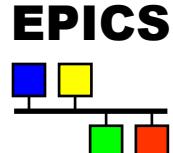
- **Use Encoder If Present (No/Yes)**

If the motor has an encoder, and the encoder is wired to the same axis as the motor output step control (normal case) the loop will be closed in the controller. Therefore, leave this as "No".

If the encoder is wired to a different input channel to the motor output, we can set this to "Yes" and close the loop in the motor record itself.

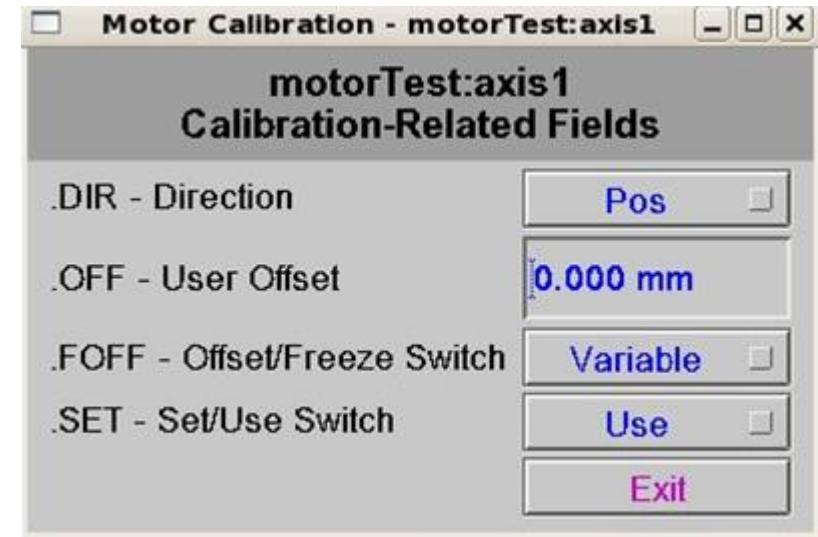


Motor Record Calibration tab



DIR/SET/FOFF – discussed earlier

OFF - Offset value

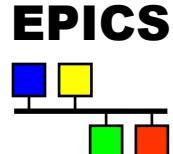


Important equation – link between user and dial coordinates:

$$\text{userVAL} = \text{DialVAL} * \text{DIR} + \text{OFFSET}$$



Motor Record Resolution tab



MRES

Physical distance moved by 1 step of the motor

SREV

Steps per resolution. Typically, 6400 for a Turbo PMAC 2 controller and 10000 for a Geobrick

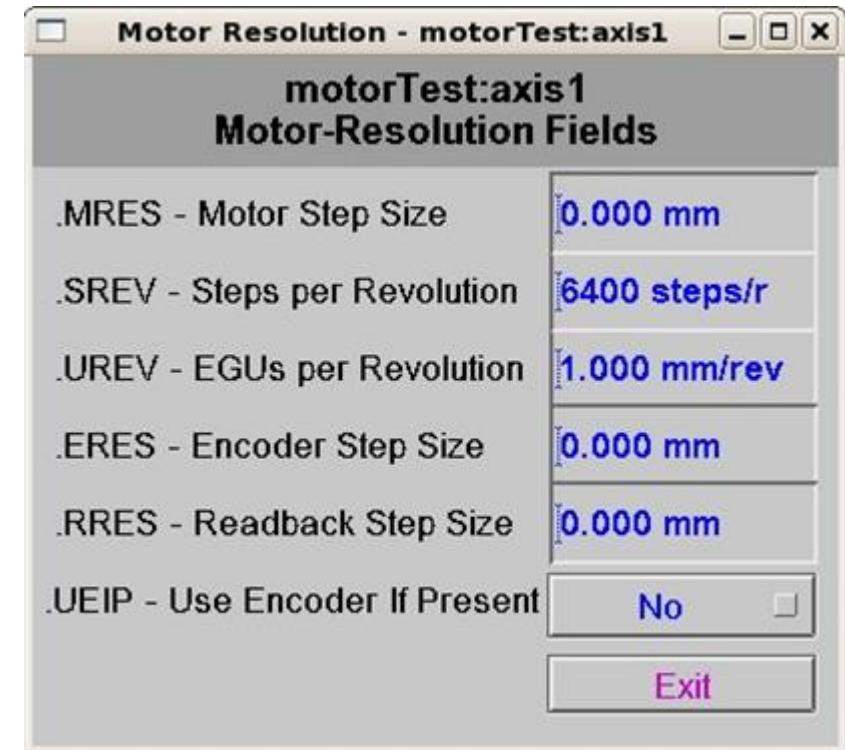
UREV

Calculated from the above two.

ERES

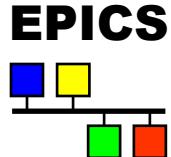
Physical distance equal to 1 count of the encoder. If there is no encoder, set this equal to MRES. Can be negative to reverse sense of limits (wiring incorrect).

RRES – Physical distance equal to 1 count or step, when steps are read back from an EPICS database link (RDBL).





Motor Record Motion tab



VMAX – Maximum velocity

VBAS – Base velocity, not used. Intent is to prevent stepper motors from exciting their resonances.

VELO – The actual velocity at which the move will be made

ACCL – The time over which VEL0 is reached from 0. Units are “seconds to velocity”

JVEL – Jog velocity.

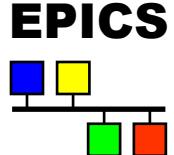
JAR – Jog acceleration.

Motor Motion - motorTest:axis1			
motorTest:axis1			
Motion Related Fields			
.VMAX - Max Velocity (EGU)	4.000 mm/sec	.SMAX (RPS)	4.000 mm
.VBAS - Base Velocity (EGU)	0.000 mm/sec	.SBAS (RPS)	0.000 rev/sec
.VELO - Velocity (EGU)	4.000 mm/sec	.S (RPS)	4.000 rev/sec
.ACCL - Seconds to Velocity	0.100 sec		
.JVEL - Jog Velocity (EGU)	4.000 mm/sec		
.JAR - Jog Acceleration (EGU)	1.000 mm/s/s		

These fields are to do with range checking and are **NOT** set by the user



Motor Record Motion tab



BDST – Backlash distance.

The distance at which to apply backlash correction.

- Motor moves past demand position or stops short
- Depends on sign

BVEL – backlash velocity

.BDST - BL Distance	0.000 mm	.SBAK (RPS) Exit
.BVEL - BL Velocity (EGU)	0.000 mm/sec	
.BACC - BL Secs to Velocity	0.100 sec	
.FRAC - Move Fraction	1.000 mm	
.RDBD - Retry Deadband (EGU)	0.000 mm	
.RTRY - Max Retry Count	0	

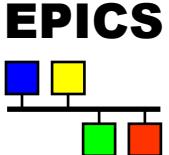
BACC – time taken to reach the backlash velocity from 0.

FRAC – fraction of backlash distance to move. Normally left as 1.

RDBD – Retry Deadband. The smallest move you can request, normally equal to the encoder resolution. If the Max. Retry Count field is set, the motor will keep trying to reach a position such that: $|(\text{Demand} - \text{Current})| < \text{RDBD}$. Used when the loop is closed in the motor record.

RTRY – Maximum number of retries. The maximum number of times, the motor will try to move to the desired position. If desired position was not reached, the field **MISS** will be set to 1

SBAK - unused



Motor Record Links tab

OUT – tells us which controller and which axis we are talking to.

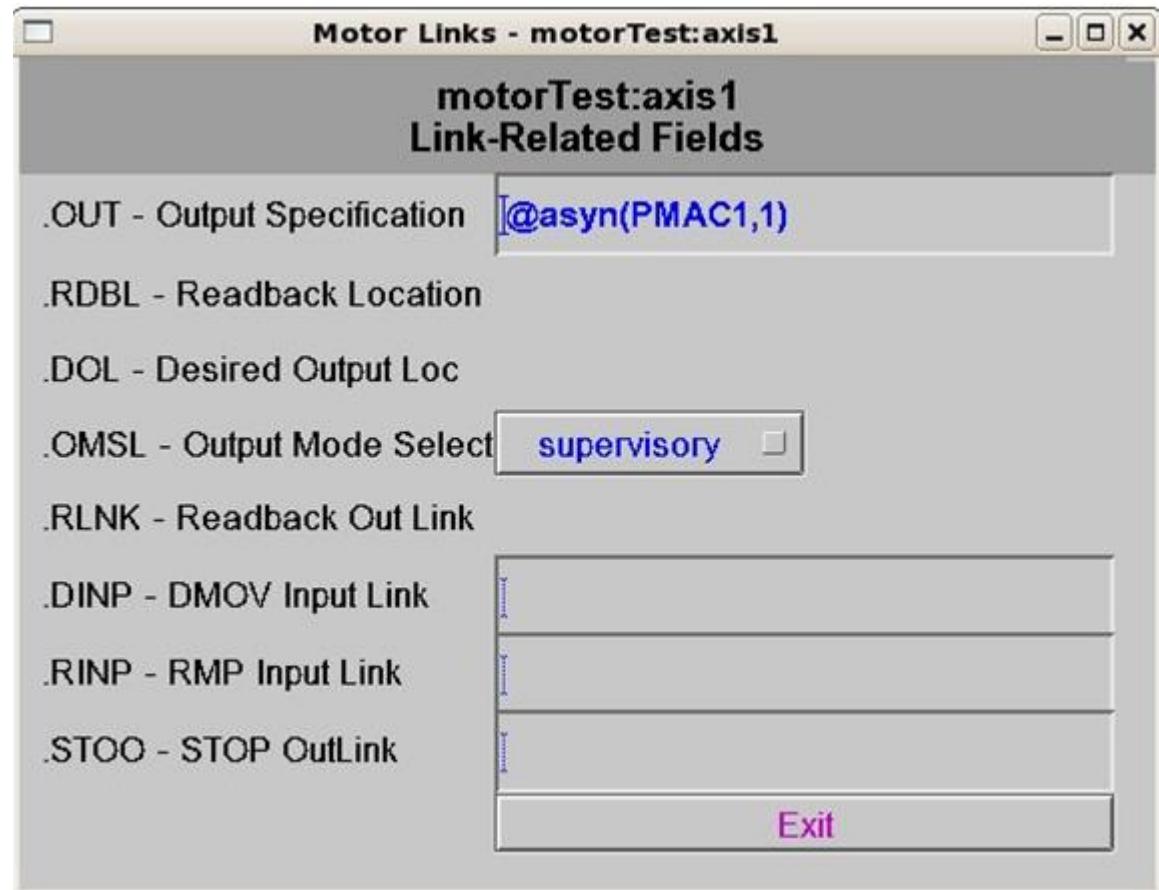
RDBL – When URIP=1 fetch the value of raw current position from this link.

DOL/OMSL – used to fetch a demand vale if OMSL=closed_loop.

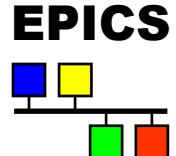
RLNK – Send current position to this location.

DINP – For soft support. Link for “Done Moving” flag.

RINP – For soft support. Link for “Raw Motor Position”.



STOO – unused in the code.



Motor Record Limits tab

HLM – Upper soft limit in user coordinates. Normally set less than physical limit.

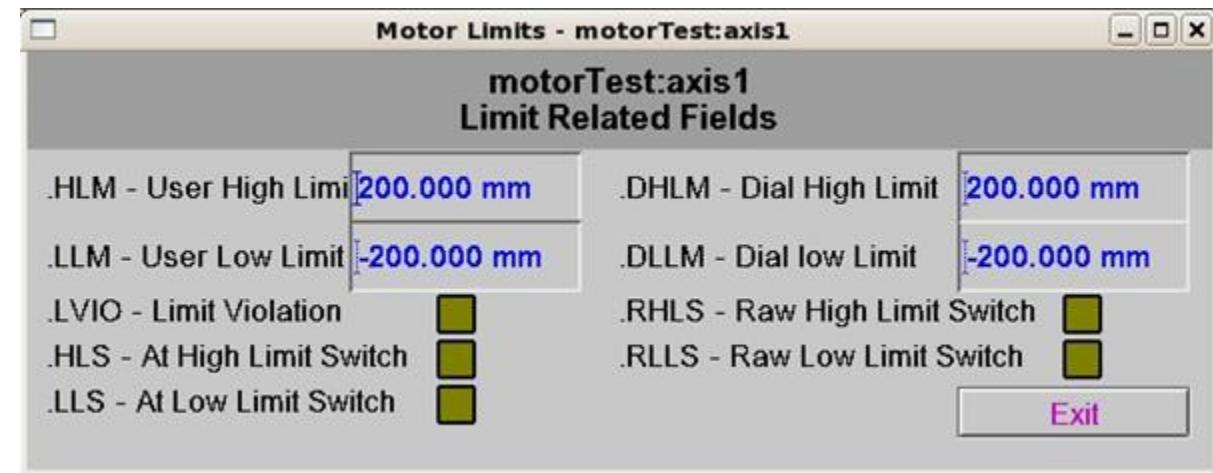
LLM – Lower soft limit in user coordinates. Normally set greater than physical limit.

DHLM – Dial high limit (takes into account offset).

DLLM – Dial low limit (takes into account offset).

DHLM = DLLM = 0

Disables limits



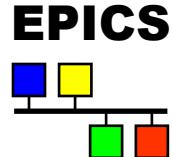
LVIO – Status bit indicating dial demand field, **DVAL**, is outside of **DHLM/DLLM**. Motor will **not** move.

HLS/LLS – Indicate that the motor is at a limit. Can be a real physical limit or a soft limit (**HLM/LLM**).

RHLS/RLLS – Indicate that the motor is at a dial high or low limit.



Motor Record Command tab



SPMG – Stop/Pause/Move/Go

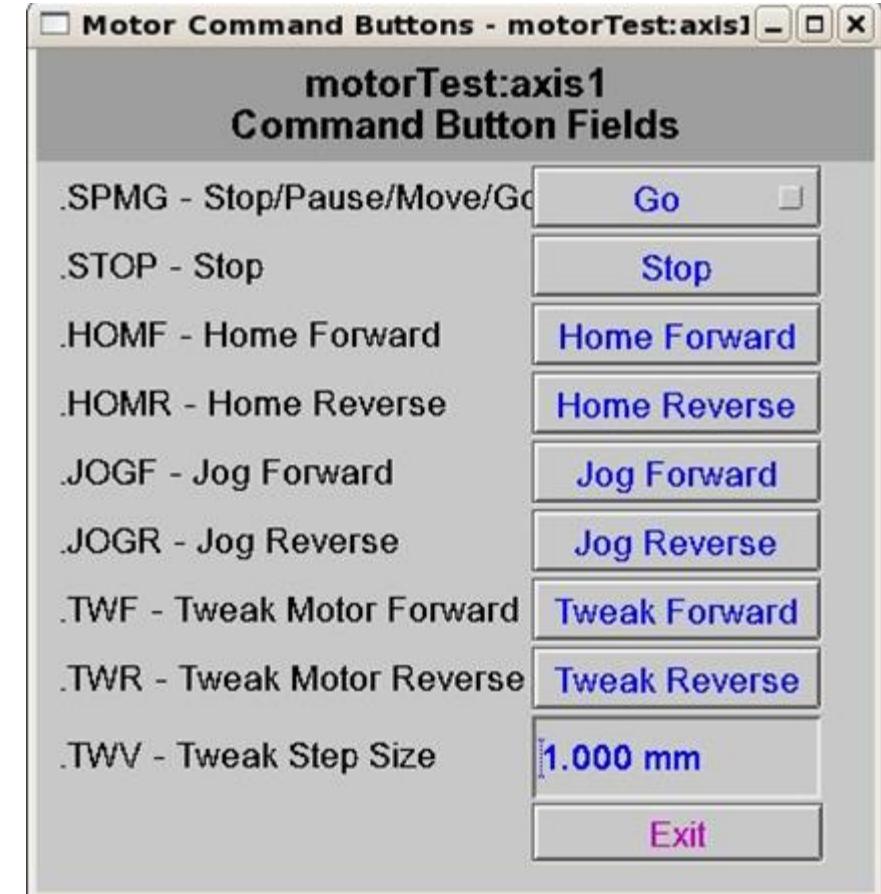
- These are 4 states.
- Not recommended to use this field.
- Normally has the value **Go**.

Go – act on all demands.

Stop – causes the motor to decelerate to a stop. The VAL field will be set equal to the RBV field and the motor will not move again while in this state.

Pause – In this state the motor will not move until this field is set to **Move**, even though a demand may have been issued.

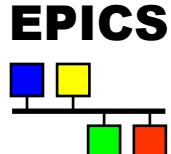
Move – Causes the motor to move to the demand position and then go back into the **Pause** state.



All other fields described earlier.



Motor Record Drive tab



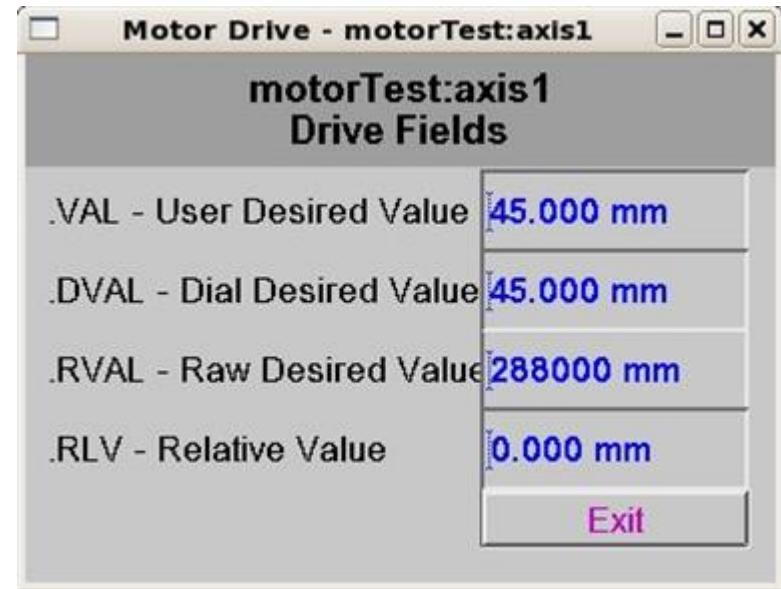
VAL – Current demand position (user coords).

DVAL – Current demand position (dial coords: takes into account offset and any direction change (**DIR**))

RVAL - Raw demand position = **DVAL/MRES**

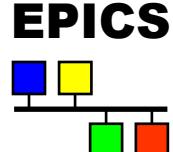
RLV – Relative value. When this field is changed, it is added to **VAL** and reset to 0. The motor record will behave as though the **VAL** field had been changed directly.

Normally, we do not use this.





Motor Record Servo tab



A collection of servo related fields.

In theory, it should be possible to use this to set the P, I, D gains of the controller.

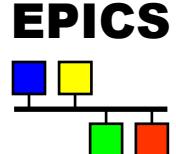
This is unsupported by the PMAC driver.

This tab, unused.





Motor Record Alarm tab



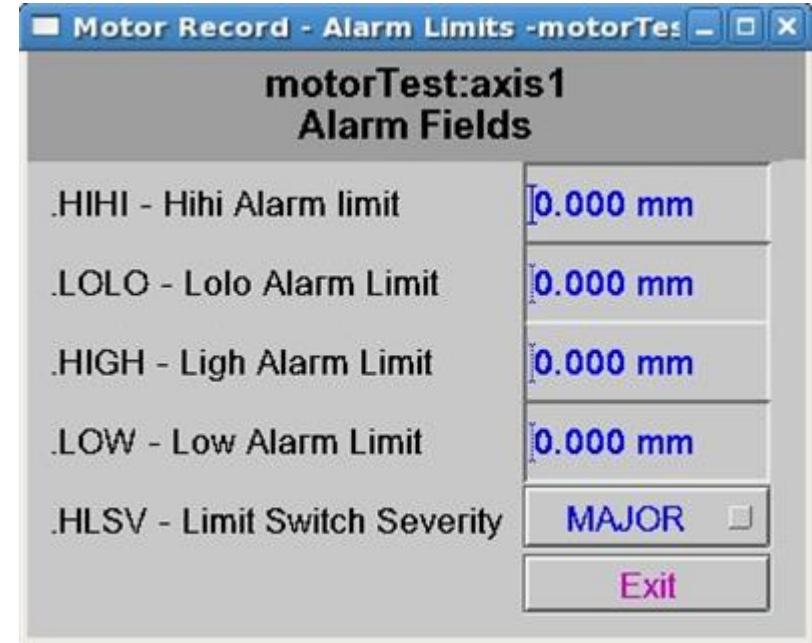
Standard EPICS alarm fields:

HIHI/LOLO/HIGH/LOW

These are unused in the motor record.

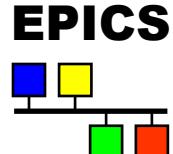
HLSV – Limit switch severity

Causes an alarm to be raised whenever the motor moves beyond either its user or dial upper or lower limits.





Motor Record Misc tab



PREC – Number of decimal places to display all fields of type double

EGU – Engineering units

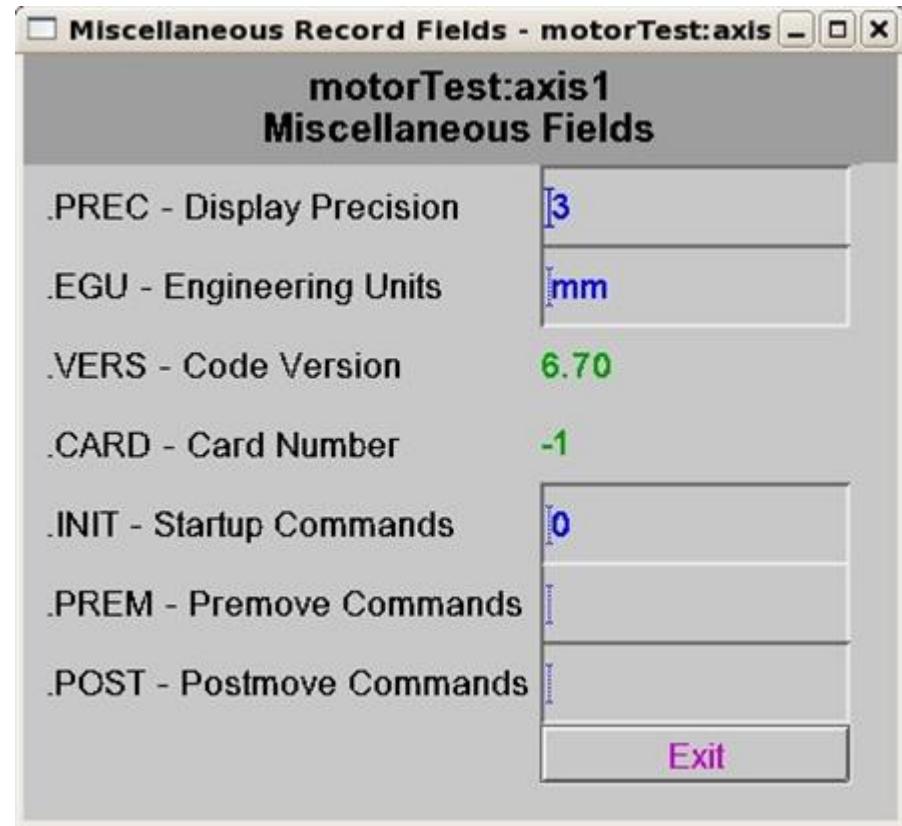
VERS – Version of the motor record

CARD – Only used for VME devices

INIT – Send this command at *iocInit*. Not supported in the PMAC driver

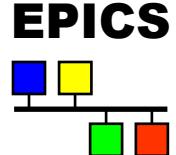
PREM – Send this command **before** any motion (any demand change). Not supported in the PMAC driver

POST – Send this command **after** any motion is completed or after any limit switch is detected. Not supported in the PMAC driver





Motor Record Status tab

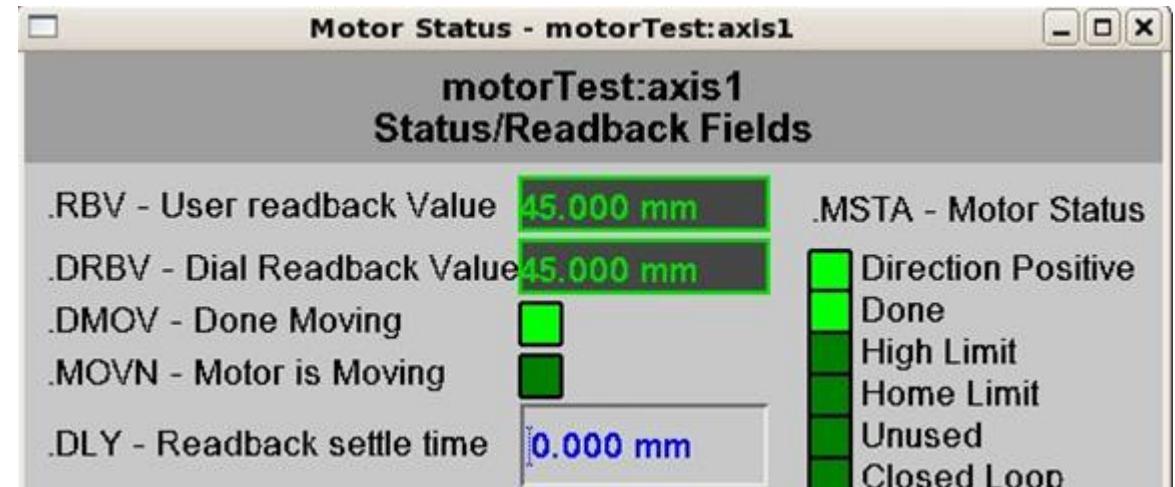


RBV – Current position in user coordinates

DRBV – Current position in dial coordinates

DMOV – Motion is complete.
DMOV is guaranteed to take the values 0 & 1 when the motor is commanded to move, even if moving to its current position.
Used by Channel Access clients.

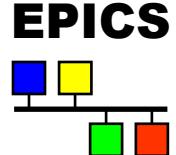
MOVN – Indicates a moving motor. Note, this is **NOT** the inverse of **DMOV** because a complex move may involve a momentary stop (reversed direction).



DLY – Time, in seconds, between the motor controller signalling done and the motor record setting **DMOV** to 1. Seldom used. Wrong units!



Motor Record Status tab



DIFF – Difference, in engineering units, between the desired motor position and the readback device's report of the current position

RDIF – Difference in “raw” units (normally steps), between the desired motor position and the readback device's report of the current position

RMP – Raw Motor Position. The current position, in motor steps or encoder counts, read from the motion controller. This depends on the configuration of the motion controller.

REP – Raw Encoder Position. As above, except if the encoder has been wired to a separate axis and we have told the software which axis this is. Then we see the encoder value here.

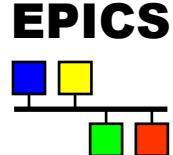
RRBV – Raw readback position. **RRBV = RMP** if **UEIP = No**. **RRBV = REP**, if **UEIP = Yes**

.DIFF - RVAL-DRBV	0.000 mm	Following Error
.RDIF - DVAL-DRBV	0	At Home
.RRBV - Raw	288000	Encoder Present
.RMP - Raw Motor Position	288000	Problem
.REP - Raw Encoder Position	288000	Moving
.TDIR - Direction of Travel	Forward	Gain Support
		Comms Error
		Low Limit
		Homed

TDIR – The current or last direction of travel



Motor Record Status tab



ATHM – This is light green when the motor is sitting at the home position

RCNT – The number of times the motor has tried to get within RDBD of the demand position



MISS – This is red when the maximum number of retries has been exceeded and the current position is **not** within RDBD of the demand position

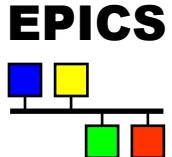
RVEL – Raw velocity (steps/second). Unused in the code.

Following Error / Max Following Error / Reset Max

These are not part of the motor record, but from a separate database template.

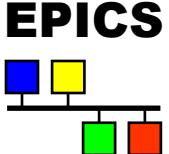


Motor Record Status bits



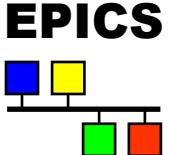
Direction Positive:	light green if last move was in positive direction.
Done:	light green if the motion is complete.
High Limit:	red if the high limit is reached.
Home Limit:	not used by PMAC support.
Unused:	
Closed Loop:	light green if in closed-loop control (#1j/).
Following Error:	red if following error has been exceeded
At Home:	light green if at home position.
Encoder Present:	light green. Always set even if no encoder
Problem:	red if unspecified problem in the controller
Moving:	light green when the motor is moving.
Gain Support:	not used by PMAC support.
Comms Error:	red when there is a controller comms error
Low Limit:	red if the low limit is reached.
Homed:	light green if the motor has been homed.

.MSTA - Motor Status	
Direction Positive	light green
Done	light green
High Limit	red
Home Limit	red
Unused	dark green
Closed Loop	light green
Following Error	red
At Home	light green
Encoder Present	light green
Problem	red
Moving	light green
Gain Support	dark green
Comms Error	red
Low Limit	red
Homed	light green
0x4103	



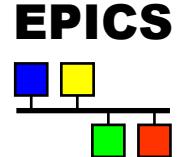
What does red text mean?

- Record has gone into a major alarm state
 - A limit has been exceeded
 - The axis has no real physical limits, but you have not told the code to stop checking
 - Add `pmacDisableLimitsCheck()` to startup script
 - The following error has been exceeded – motion is killed
 - Is there an obvious physical reason
 - Try lowering the velocity (VELO)
 - Try increasing the following error, with caution!
 - Who are you gonna call - Call the motion team!



Why isn't my motor moving?

- Many possibilities!
 - Are we outside of the soft limits or demanding a position beyond these limits (Limits tab)?
 - Is there a virtual motor sitting on top of us and its limits are stopping us?
Common examples are:
 - Slits: gap and centre
 - Mirrors: roll, pitch and yaw
 - Are we in the "Stop" or "Pause" state (Commands tab, SPMG)
 - Is the "Set/Use" field (SET) equal to "Set"?
 - Is our velocity (VELO) zero?
 - Are you trying to move less than RDBD?



areaDetector

A module for EPICS area detector support

Andy Foster

(based on a presentation by Mark Rivers,
University of Chicago
Advanced Photon Source)

areaDetector Talk Outline

- Motivation & goals for areaDetector module
- Overview of architecture, data structures used
- Detector drivers in use at Diamond
- Plugins
 - Commonly used plugins
 - Plugins for file writing
- Data viewers
- Using the IOC Builder with areaDetector IOC's
- Useful links for further reading

areaDetector - Motivation

- 2-D detectors are essential components of synchrotron beamlines
 - Sample viewing cameras
 - X-ray diffraction and scattering detectors
 - X-ray imaging etc.
- EPICS is used on many beamlines all over the World:
 - Diamond, Australia, APS, SLAC, NSLS-II, SSRF (Shanghai), etc.
- Clear advantages to an architecture that can be used on any detector, re-using many software components
- Providing EPICS control allows any higher-level client to control the detector and access the data
 - CSS, EDM, GDA, SPEC, Python scripts etc.
 - The client needs only to know how to talk to EPICS, not the details of the detector

areaDetector - Goals

- Drivers for many detectors popular at synchrotron beamlines
 - Handle detectors ranging from >500 frames/second to <1 frame/second
- There are basic parameters which all detectors have:
 - Exposure time, start acquisition, gain,...
 - Allows generic clients to be used for many applications
- Easy to implement new detector
 - Single device-driver C++ file to write (asynPortDriver)
- Easy to add detector-specific features
 - Driver understands additional parameters beyond those in the basic set
- EPICS-independent at lower layers (interface to detector)
- Plug-ins to add capabilities like:
 - region-of-interest calculation, statistics, overlays, file saving, etc.
 - Detector independent, work with all drivers
 - Run below the EPICS database and Channel Access layers for highest performance

areaDetector – architecture

- “Pipelining Acquisition and Processing Framework”
- Driver
 - Acquires data and passes it into the framework
 - A producer in the framework
- Plugin
 - Processes data from a producer – i.e. a consumer
 - Data arrives in the form of N-dimensional arrays with associated meta-data (attributes)
 - Can effectively act as a filter: Outputs a processed N-dimensional array (i.e. a consumer and a producer)
- Typical areaDetector application
 - one Driver followed by a chain of Plugins
- Plugins, generally, do not alter data
 - A ptr to the data is passed through the plugin chain.
- If they need to alter data, they should make a copy of the data first:
 - “proc” and “overlay” alter data

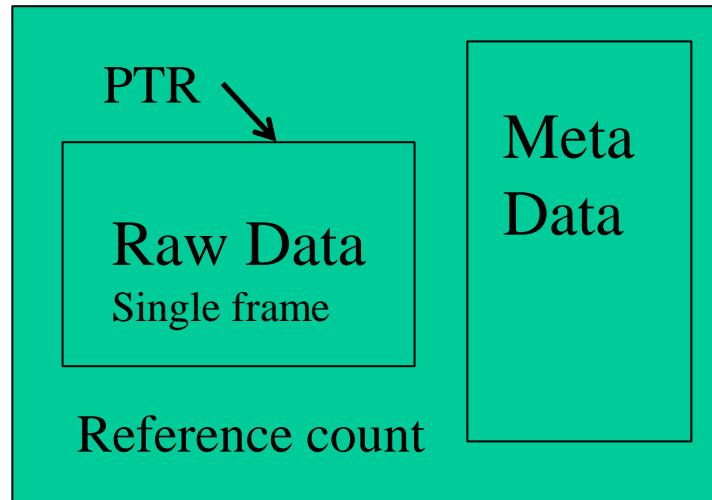
areaDetector: Data Structures

NDArray N-Dimensional array.

- Everything is done in N-dimensions (up to 10), rather than 2.
- This is needed even for 2-D detectors to support color.
- This is what plug-ins callbacks receive from device drivers.

NDAttribute

- Each NDArray has a list of associated attributes (metadata) that travel with the array through the processing pipeline.
- Attributes can come from driver parameters or any EPICS PV.
- E.g. can store motor positions, temperature, ring current, etc. with each frame.



NDArrayPool

- Allocates NDArray objects from a freelist
- Plugins access in readonly mode, increment reference count
- Eliminates need to copy data when sending it to plugins.

R3-10 (January '21) Organization

- Hosted at <https://github.com/areaDetector> project
- 59 separate git repositories
- Can be released independently

ADCore

Core module
Base classes, plugins,
documentation

ADSupport

Third party libraries in
source code to build for
Windows: HDF5,
GraphicsMagick and
others

ADAravis

Driver for GenICam
cameras
Using the Aravis
library on Linux

ADPilatus

Pilatus driver

...

Detector drivers in use at Diamond

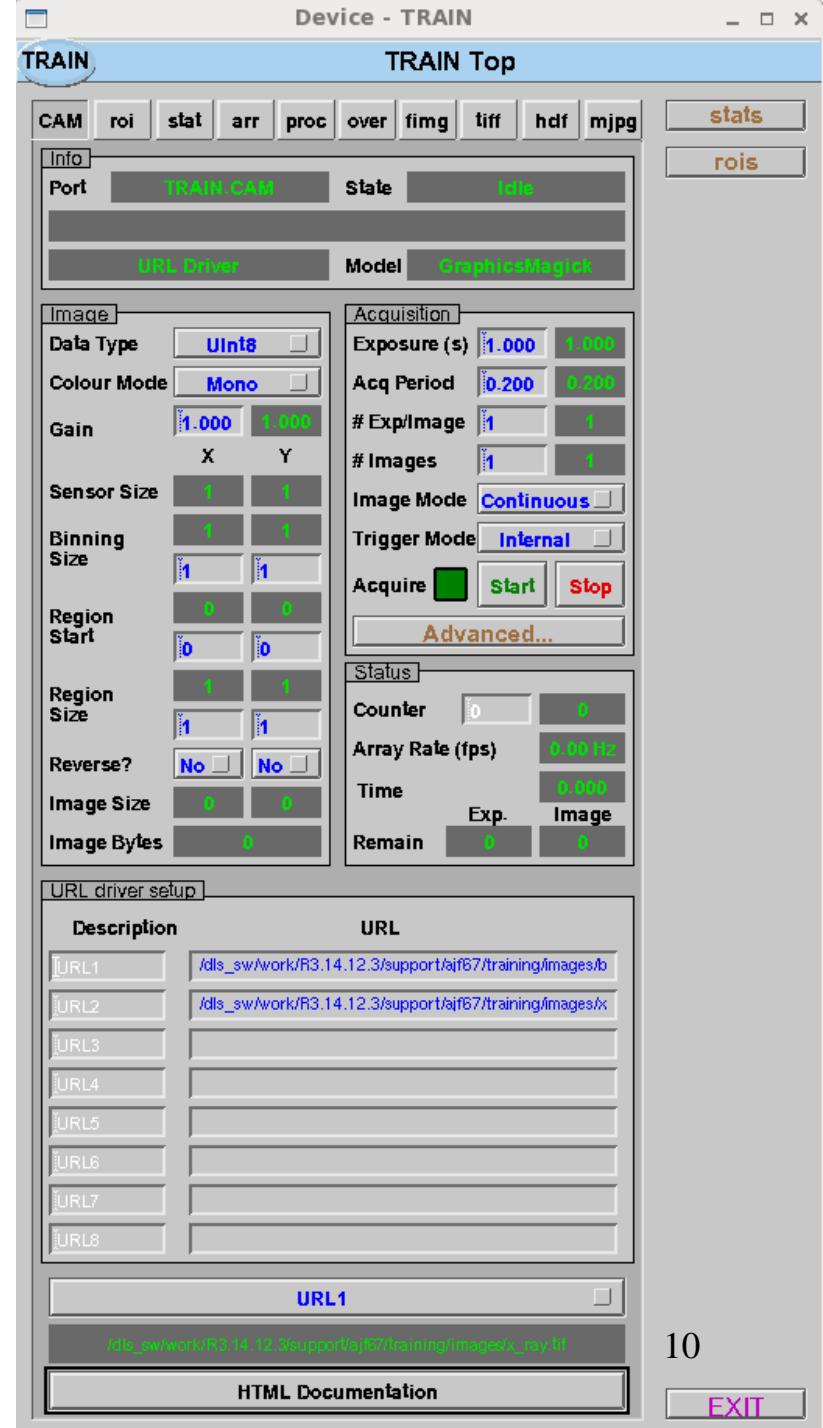
- ADDriver (in ADCore)
 - Base C++ class from which detector drivers derive.
 - Handles details of EPICS interfaces, and other common functions.
- Simulation driver (in ADCore)
 - Produces calculated images up to very high rates.
 - Implements nearly all basic parameters, including color.
 - Useful as a model for real detector drivers, and to test plugins and clients.
- Generic GenICam driver (ADAravis)
 - Should work with any GenICam compliant camera (diagnostic cameras)
 - Controlled using the Aravis library under Linux.
- Prosilica driver (ADProsilica)
 - Gigabit Ethernet cameras, mono and color
 - High resolution, high speed, e.g. 1360x1024 at 30 fps = 40MB/second.
 - Controlled via calls to vendor PvAPI C library.

Detector drivers in use at Diamond (contd)

- Pilatus driver (ADPilatus)
 - Pilatus pixel-array detectors from Dectris. Controlled via sockets to vendor camserver application
- URL driver (ADURL)
 - Driver to display images from any URL. Works with Web cameras, Axis video servers, static images, etc. Controlled by calls to GraphicsMagick library
- HDF5 driver (simHDF5Detector)
 - Driver to “replay” dataset from a HDF5 file. Much faster than simDetector
 - Developed by Alan Greer (OSL).
- PSL driver (ADPSL)
 - Photonic Science Limited detectors
 - Controlled via socket connection to their Python PSLViewer server
- Andor & Andor3 drivers (ADAndor, ADAndor3)
 - Driver for Andor CCD cameras (calls to V2 of their “C” SDK)
 - Driver for Andor sCMOS cameras (calls to V3 of their SDK)

URL Driver

- Driver that can read images from any URL.
- Can be used with Web cameras and Axis video servers.
- Uses GraphicsMagick to read the images, and can thus handle a large number of image formats (JPEG, TIFF, PNG, etc.).



Plugins

- Designed to perform real-time processing of data, running in the EPICS IOC (not over EPICS Channel Access)
- Receive ptr to NDArray object in a callback from driver or other plugins
 - areaDetector core code takes care of notifying all plugins which are interested when a callback is made.
- Each plugin has a queue whose size is configured in the IOC startup script.
- Plugins can execute in their own threads (non-blocking) or in the callback thread (blocking)
 - If non-blocking then NDArray ptr's are put on the queue
 - If the queue fills up, images are dropped!
 - While NDArrays are queued, they cannot be freed back to the pool
 - If executing in callback thread, no queuing, but slows device driver
 - Because processing cannot continue along the chain, while this thread is running.

Plugins (continued)

- File writing plugins such as HDF, TIFF **must** have large queues
 - do not want to drop images while waiting for the file system. Typical value: 5000 objects.
- ffmpegServer only has a queue of 2
 - It doesn't matter if we drop images here, we only want to see the current image.
- At runtime, each plugin allows control of:
 - **Enabling/disabling**
 - **Execution (throttling) rate** i.e. only execute at a limited rate e.g. detector can be saving data to disk at full speed, but images can be posted to EPICS at a reduced rate
 - **Changing data source** for NDArray callbacks to another driver or plugin
- Some plugins are also sources of NDArrays, as well as consumers:
 - Allows several, simultaneous, threads of processing to occur i.e. a producer can feed more than one consumer.

Commonly used Plugins

- **NDPlugInStdArrays**
 - Receives NDArrays (images) from device drivers, converts to standard arrays, e.g. arrays which can be stored in waveform records.
 - This plugin is what EPICS channel access viewers normally talk to.
- **NDPluginROI**
 - Performs region-of-interest calculations
 - Select a sub-region.
 - Options to: bin, reverse in either direction, convert data type
 - Divide the array by a scale factor, which is useful for avoiding overflow when binning.
- **NDPluginTransform**
 - Performs geometric operations (rotate, mirror in X or Y, etc.)
- **NDPluginStats**
 - Calculates basic statistics on an array (min, max, sigma)
- **NDPluginProcess**
 - Does arithmetic processing on arrays: Offset and scale.
 - Background subtraction. Conversion to a different output data type.

Commonly used Plugins (continued)

- **NDPluginOverlay**
 - Adds graphic overlays to an image: text, cross, rectangle, ellipse.
 - Can be used to display ROIs, multiple cursors, user-defined boxes, text.
- **ffmpegServer**
 - MJPEG server that allows viewing images in a Web browser.
 - Puts compressed images on the network, greatly reducing bandwidth compared to uncompressed channel access arrays.
- **adPython**
 - Users can write frame processing scripts in python and load the script into the plugin chain at runtime. Sample centering calculations
- **NDPluginCircularBuffer**
 - Buffers NDArrays in a circular buffer.
 - Outputs the arrays on receipt of a trigger, either as PV or NDArray attribute. Supports pre-trigger and post-trigger samples (Alan Greer, OSL)

Diamond EDM Client

- One base screen per driver and plugin
- Top level screen with tabs can be generated by IOC builder



Plugins: NDPluginFile

- Base class from which actual file plugins are derived e.g. tiff, hdf5
- Save NDArrays to disk
- 3 modes of file saving:
 - **Single** array per disk file (each frame in a different file)
 - **Capture** N arrays in memory, write to disk either multiple files or as a single large file (for file formats that support this.)
 - **Stream** arrays to a single large disk file
- For file formats that support it, stores not just NDArray data but also NDAttributes
 - NDAttribute is a key, value pair – the key is a string, value is a scalar
 - Source of the NDAttribute can be an EPICS PV or a driver parameter (from parameter library)
 - NDAttributes are defined in an XML file which can be loaded into the IOC

Plugins: NDPluginFile

- Common file formats currently supported:
 - **NDFileHDF5**
 - Writes HDF5 files with the native HDF5 API. Supports 3 types of compression.
 - Supports using an XML file to define the layout and placement of NDArrays and NDAttributes in the HDF5 file
 - **NDFileTIFF**
 - Supports any NDArray data type
 - Stores NDAttributes as ASCII user tags. One array per file
 - **NDFileJPEG**
 - With compression control. One array per file
 - **NDFileMagick**
 - Uses GraphicsMagick to write files, and can write in dozens of file formats, including JPEG, TIFF, PNG, PDF, etc.
 - **NDFileNexus**
 - Has been commonly used in Neutron/X-ray facilities.
 - Can be replicated by using HDF5 with an appropriate XML layout.

Viewers

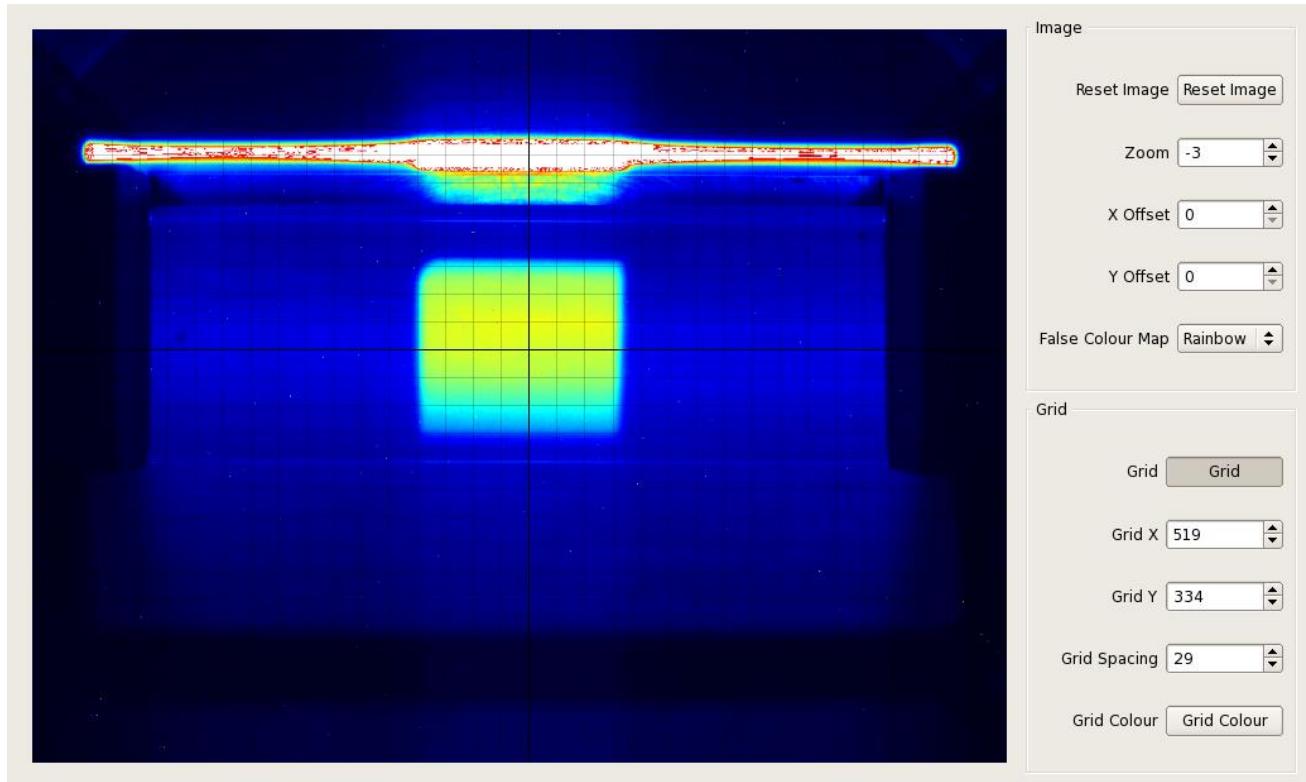
- areaDetector allows generic viewers to be written that receive images as EPICS waveform records over Channel Access
- Current viewers include:
 - **ImageJ** is a very popular image analysis program, written in Java. Plugin is EPICS_AD_Viewer.
 - **IDL** (plugin EPICS_AD_Display).
 - **ffmpegViewer** high-performance Qt-based viewer for the mjpg plugin. The mjpg plugin contains ffmpegServer.

ffmpegViewer at DLS

- ffmpegServer module contains
 - ffmpeg compression
 - http streaming
 - QT viewer of ffmpeg compressed stream
- Start from commandline

```
ffmpegViewer  
Usage: ffmpegViewer [options] <mjpg_url> [<CA prefix for grid>]
```

-h	Show this help message and quit
-d	Do not show docking controls on right of player window
-f	Fallback mode, don't try to use xvideo



Use the builder!

- The IOC builder automates the building of
 - Database substitutions file, EDM top level screen, Startup script
- areaDetector IOC's contain a large number of database templates
 - At least two per plugin and driver
- Many configure functions to define in the startup script
 - Several different parameters have to be defined
 - **asyn port names** link plugins and drivers together
- Several embedded EDM screen need to be glued together to match the selection of plugins and drivers
- XEB is an XML editor to help remind us of all the various plugins and their parameters
- The DLS “adUtil” module has `gdaPlugins.xml` which contains all the “standard” plugins that GDA requires to operate an areaDetector

Use the builder!

File Edit View Search Tools Documents Help

Open Save Undo Cut Copy Paste Find Find Next Find Previous

BL18I-EA-IOC-07_expanded.substitutions

```

4# TIMEOUT      Asyn timeout
5# NDARRAY_PORT Input Array Port
6# NDARRAY_ADDR Input Array Address
7# Enabled      Plugin Enabled at startup?
8# File $(ADCORE)/db/NDPosPlugin.template
9{
10 pattern { P, R, PORT, ADDR, TIMEOUT, NDARRAY_PORT, NDARRAY_ADDR, Enabled }
11 { "BL18I-EA-DET-06", "POS", "EA4.CMOS.POS", "0", "1", "EA4.CMOS.CAM", "0", "1" }
12 }
13
14 # Macros:
15 # P           Device Prefix
16 # R           Device Suffix
17 # PORT        Asyn Port name
18 # TIMEOUT     Timeout
19 # ADDR        Array Port address
20 # NDARRAY_PORT Input Array Port
21 # NDARRAY_ADDR Input Array Address
22 # Enabled      Plugin Enabled at startup?
23 file $(ADCORE)/db/NDROI.template
24 {
25 pattern { P, R, PORT, TIMEOUT, ADDR, NDARRAY_PORT, NDARRAY_ADDR, Enabled }
26 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi", "1", "0", "EA4.CMOS.POS", "0", "1" }
27 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi.roi1", "1", "0", "EA4.CMOS.POS", "0", "1" }
28 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi.roi1", "1", "0", "EA4.CMOS.POS", "0", "1" }
29 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi.roi3", "1", "0", "EA4.CMOS.POS", "0", "1" }
30 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi.roi4", "1", "0", "EA4.CMOS.POS", "0", "1" }
31 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi.roi5", "1", "0", "EA4.CMOS.POS", "0", "1" }
32 { "BL18I-EA-DET-06", "ROI1", "EA4.CMOS.roi.roi6", "1", "0", "EA4.CMOS.POS", "0", "1" }
33 }
34
35 # Macros:
36 # P           Device Prefix
37 # R           Device Suffix
38 # PORT        Asyn Port name
39 # TIMEOUT     Timeout
40 # ADDR        Array Port address
41 # NDARRAY_PORT Input Array Port
42 # NDARRAY_ADDR Input Array Address
43 # Enabled      Plugin Enabled at startup?
44 file $(ADCORE)/db/NDStats.template
45 {
46 pattern { P, R, PORT, TIMEOUT, ADDR, XSIZE, YSIZE, HIST_SIZE, NCHANS, NDARRAY_PORT, NDARRAY_ADDR, Enabled }
47 { "BL18I-EA-DET-06", "STAT", "EA4.CMOS.stat", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.POS", "0", "1" }
48 { "BL18I-EA-DET-06", "STAT1", "EA4.CMOS.stat.stat1", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.roi.roi1", "0", "0" }
49 { "BL18I-EA-DET-06", "STAT2", "EA4.CMOS.stat.stat2", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.roi.roi1", "0", "0" }
50 { "BL18I-EA-DET-06", "STAT3", "EA4.CMOS.stat.stat3", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.roi.roi1", "0", "0" }
51 { "BL18I-EA-DET-06", "STAT4", "EA4.CMOS.stat.stat4", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.roi.roi1", "0", "0" }
52 { "BL18I-EA-DET-06", "STAT5", "EA4.CMOS.stat.stat5", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.roi.roi1", "0", "0" }
53 { "BL18I-EA-DET-06", "STAT6", "EA4.CMOS.stat.stat6", "1", "0", "4320", "2048", "256", "20000", "EA4.CMOS.roi.roi1", "0", "0" }
54 }
55
56 # Macros:
57 # P           Device Prefix
58 # R           Device Suffix
59 # PORT        Asyn Port name
60 # TIMEOUT     Timeout
61 # ADDR        Array Port address
62 # XSIZE       Maximum size of X histograms (e.g. 1024)
63 # YSIZE       Maximum size of Y histograms (e.g. 768)
64 # HIST_SIZE   Maximum size of Pixel binning histogram (e.g. 256 for Int8)
65 # NCHANS      Number of elements in the waveforms which accumulate past statistics
66 # NDARRAY_PORT Input Array Port
67 # NDARRAY_ADDR Input Array Address
68 # Enabled      Plugin Enabled at startup?
69 file $(ADCORE)/db/NDStdArrays.template
70 {
71 pattern { P, R, PORT, TIMEOUT, ADDR, TYPE, FTVL, NELEMENTS, NDARRAY_PORT, NDARRAY_ADDR, Enabled }
72 { "BL18I-EA-DET-06", "ARR", "EA4.CMOS.arr", "1", "0", "Int16", "USHORT", "8847360", "EA4.CMOS.roi", "0", "0" }
73 }
74
75 # Macros:
76 # P           Device Prefix
77 # R           Device Suffix
78 # PORT        Asyn Port name
79 # TIMEOUT     Timeout
80
81 # Macros:
82 # P           Device Prefix
83 # R           Device Suffix
84 # PORT        Asyn Port name
85 # TIMEOUT     Timeout
86 # ADDR        Array Port address
87 # TYPE        Asyn Type e.g. Int32
88 # FTVL        Format, e.g. Int
89 # NELEMENTS   Number of elements
90 # NDARRAY_PORT Input Array Port
91 # NDARRAY_ADDR Input Array Address
92 # Enabled      Plugin Enabled at startup?
93 file $(ADCORE)/db/NDProcess.template
94
95 pattern { P, R, PORT, TIMEOUT, ADDR, TYPE, FTVL, NELEMENTS, NDARRAY_PORT, NDARRAY_ADDR, Enabled }
96 { "BL18I-EA-DET-06", "ARR", "EA4.CMOS.arr", "1", "0", "Int16", "USHORT", "8847360", "EA4.CMOS.roi", "0", "0" }
97 }
98
99 # Macros:
100 # P           Device Prefix
101 # R           Device Suffix
102 # PORT        Asyn Port name
103 # TIMEOUT     Timeout

```

Plain Text ▾ Tab Width: 4 ▾ Ln 6, Col 1 INS

File Edit View Search Tools Documents Help

Open Save Undo Cut Copy Paste Find Find Next Find Previous

BL18I-EA-IOC-07_expanded.substitutions

stBL18I-EA-IOC-07.src

```

1 # This file was automatically generated on Thu 08 Oct 2015 10:15:59 BST from
2 # source: /dls_sw/work/R3.14.12.3/support/mapping/ADPSL/etc/makelocs/BL18I-EA-IOC-07.xml
3 #
4 # *** Please do not edit this file: edit the source file instead. ***
5 #
6 cd "$(INSTALLED)"
7
8 epicsEnvSet "EPICS_TS_MIN_WEST", "0"
9
10
11 # Loading libraries
12 #
13
14 # Device initialisation
15 #
16
17 cd "$(INSTALLED)"
18
19 dbLoadDatabase "db/BL18I-EA-IOC-07.dbd"
20 BL18I_EA_IOC_07_registerRecordDeviceDriver(pdbbase)
21
22 drvAsynIPPortConfigure("EA4.CMOS.CAM.ip", "10.10.10.100:50000", 100, 0, 0)
23
24 # PSLConfig(portName, serverPort, maxBuffers, maxMemory)
25 PSLConfig("EA4.CMOS.CAM", "EA4.CMOS.CAM.ip", 10, 176947200)
26
27 NDPSPluginConfigure("EA4.CMOS.POS", 10, 0, "EA4.CMOS.CAM", 0, 50, 88473600, 0, 0)
28 NDPSPluginConfigure("EA4.CMOS.POS", 10, 0, "EA4.CMOS.CAM", 0, 50, 88473600, 0, 0)
29
30 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
31 NDROIConfigure("EA4.CMOS.roi", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
32
33 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
34 NDROIConfigure("EA4.CMOS.roi1", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
35
36 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
37 NDROIConfigure("EA4.CMOS.roi2", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
38
39 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
40 NDROIConfigure("EA4.CMOS.roi3", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
41
42 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
43 NDROIConfigure("EA4.CMOS.roi4", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
44
45 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
46 NDROIConfigure("EA4.CMOS.roi5", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
47
48 # NDROIConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
49 NDROIConfigure("EA4.CMOS.roi6", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
50
51 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
52 NDStatsConfigure("EA4.CMOS.stat", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
53
54 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
55 NDStatsConfigure("EA4.CMOS.stat.stat1", 16, 0, "EA4.CMOS.roi.roi1", 0, 0, 0)
56
57 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
58 NDStatsConfigure("EA4.CMOS.stat.stat2", 16, 0, "EA4.CMOS.roi.roi1", 0, 0, 0)
59
60 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
61 NDStatsConfigure("EA4.CMOS.stat.stat3", 16, 0, "EA4.CMOS.roi.roi1", 0, 0, 0)
62
63 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
64 NDStatsConfigure("EA4.CMOS.stat.stat4", 16, 0, "EA4.CMOS.roi.roi1", 0, 0, 0)
65
66 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
67 NDStatsConfigure("EA4.CMOS.stat.stat5", 16, 0, "EA4.CMOS.roi.roi1", 0, 0, 0)
68
69 # NDStatsConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
70 NDStatsConfigure("EA4.CMOS.stat.stat6", 16, 0, "EA4.CMOS.roi.roi1", 0, 0, 0)
71
72 # NDStdArraysConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
73 NDStdArraysConfigure("EA4.CMOS.arr", 2, 0, "EA4.CMOS.POS", 0, 0, 0)
74
75 # NDProcessConfigure(portName, queueSize, blockingCallbacks, NDArrayPort, NDArrayAddr, maxBuffers, maxMemory)
76 NDProcessConfigure("EA4.CMOS.proc", 16, 0, "EA4.CMOS.POS", 0, 0, 0)
77

```

Plain Text ▾ Tab Width: 4 ▾ Ln 1, Col 1 INS

Use the builder!

The screenshot shows the ADBuilder application interface with three separate windows, each displaying a table configuration:

- Top Window (ADPSL.PSL):** Shows a table with columns X, #, PORT, P, R, TIMEOUT, ADDR, and IP_PORT. One row is present: EA4.CMOS.CAM, EA4.CMOS.CAM, BL18I-EA-DET-06, :CAM:, 1, 0, 10.10.10.100:50000.
- Middle Window (adUtil.gdaPlugins):** Shows a table with columns X, #, name, CAM, P, PORTPREFIX, and PLUGINQUEUE. One row is present: gdapugins, gdapugins, EA4.CMOS.POS, BL18I-EA-DET-06, EA4.CMOS, 16, 0.
- Bottom Window (EPICS_BASE.StartupCommand):** Shows a table with columns X, #, name, command, post_init, and at_end. Fourteen rows are listed, each representing a command related to the BL18I-EA-DET-06 camera setup.

X	#	name	command	post_init	at_end
[row 1]			asynOctetSetOutputEos("EA4.CMOS.CAM.ip", 0, "\n")	False	True
[row 2]			asynSetTraceIOMask("EA4.CMOS.CAM.ip", 0, 1)	False	True
[row 3]			asynSetTraceMask("EA4.CMOS.CAM.ip", 0x3F)	False	True
[row 4]			dbpf "BL18I-EA-DET-06:CAM:CameraName 5"	True	True
[row 5]			dbpf "BL18I-EA-DET-06:FIMG:fileTemplate", "%s/%s_%d.avi"	True	True
[row 6]			dbpf "BL18I-EA-DET-06:HDF5:fileTemplate", "%s/%s_%d.h5"	True	True
[row 7]			dbpf "BL18I-EA-DET-06:HDF5:AutoIncrement", "Yes"	True	True
[row 8]			dbpf "BL18I-EA-DET-06:HDF5:fileWriteMode", "Stream"	True	True
[row 9]			dbpf "BL18I-EA-DET-06:HDF5:LazyOpen", "Yes"	True	True
[row 10]			dbpf "BL18I-EA-DET-06:HDF5:enableCallbacks", "Enable"	True	True
[row 11]			dbpf "BL18I-EA-DET-06:TIFF:fileTemplate", "%s/%s_%d.h5"	True	True
[row 12]			dbpf "BL18I-EA-DET-06:TIFF:AutoIncrement", "Yes"	True	True
[row 13]			dbpf "BL18I-EA-DET-06:TIFF:fileWriteMode", "Stream"	True	True
[row 14]			dbpf "BL18I-EA-DET-06:TIFF:LazyOpen", "Yes"	True	True

xeb /dls_sw/work/R3.14.12.7/support/ajf67/EPICSEExercises/ADTraining/etc/makeIocs/exUser1.xml

areaDetector links

The docs:

<http://cars9.uchicago.edu/software/epics/areaDetectorDoc.html>

The sources and collaboration, issues, pull requests, etc:

www.github.com/areaDetector

The wiki: www.github.com/areaDetector/areaDetector/wiki

NSLS-II EPICS Lecture series:

<https://www.bnl.gov/ps/epics>