EPICS Documentation

EPICS

Jan 17, 2024

CONTENTS

The **Experimental Physics and Industrial Control System** (**EPICS**) comprises a set of software components and tools that can be used to create distributed control systems. EPICS provides capabilities that are typically expected from a distributed control system:

- Remote control & monitoring of facility equipment
- Automatic sequencing of operations
- Facility mode and configuration control
- Management of common time across the facility
- Alarm detection, reporting and logging
- Closed loop (feedback) control
- Modeling and simulation
- Data conversions and filtering
- Data acquisition including image data
- Data trending, archiving, retrieval and plotting
- Data analysis
- Access security (basic protection against unintended manipulation)

EPICS can scale from very big to very small systems. Big systems have to be able to transport and store large amounts of data, be robust and reliable but also failure-tolerant. Failure of a single component should not bring the system down. For small installations it has to be possible to set up a control system without requiring complicated or expensive infrastructure components.

For modern applications, management of data is becoming increasingly important. It shall be possible to store acquired operational data for the long term and to retrieve it in the original form. EPICS provides the tools to achieve this and to tailor the data management to the needs of the facility.

One of the most appreciated aspects of EPICS is the lively collaboration that is spread around the globe. Members of the collaboration are happy to help other users with their issues and to discuss new ideas.

CHAPTER

ONE

HOW THIS DOCUMENTATION IS ORGANIZED

Each page is labeled by the intended audienece. You may also directly use related links to see documents which match you the most.

Tags: beginner user developer advanced

1.1 Getting started with EPICS

Tags: beginner

1.1.1 System components

Broadly speaking, the EPICS toolset enables creation of servers and client applications. Servers provide access to data, reading or writing, locally or over a network. Reading and writing is often done to and from hardware connected to physical components, however data can also be produced or used elsewhere. Physical I/O, however is the central task of any control system, including EPICS.

Clients can display, store and manipulate the data. Client software ranges from (graphical and command line) user interface tools to powerful services for data management.

The basic components of an EPICS-based control system are:

IOC, the Input/Output Controller. This is the I/O server component of EPICS. Almost any computing platform that can support EPICS basic components like databases and network communication can be used as an IOC. One example is a regular desktop computer, other examples are systems based on real-time operating systems like vxWorks or RTEMS and running on dedicated modular computing platforms like MicroTCA, VME or CompactPCI. EPICS IOC can also run on low-cost hardware like RaspberryPi or similar.

CWS, or Client WorkStation. This is a computer that can run various EPICS tools and client applications; typical examples are user interface tools and data archiving. CWS can be desktop computer, a server machine or similar, and is usually running a "regular" (as opposed to real-time) operating system like Linux, Windows or MacOS.

LAN Local Area Network. This is just a standard Ethernet-based (or wireless) communication network that allows the IOCs and CWS's to communicate.

A simple EPICS control system can be composed of one or more IOCs and Client WorkStations that communicate over a LAN (Figure 1). Separation of clients and servers makes configuration of the systems easier and also makes the system more robust. Clients and servers can be added to and removed from the system without having to stop the operation.

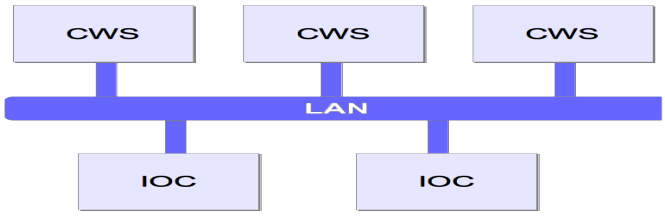


Figure 1. A simple EPICS control system structure.

In addition to these basic components of a "classical" EPICS control system, it is also possible to implement servers (aka services) for data that are not "process I/O" (real-time values from a controlled process) or attached to hardware. These other services can for example provide configuration or calibration data, or computing services like particle beam modeling. Since all the services "speak" the same protocol and exchange the same type of data structures, the data source is transparent to the client software (i.e., you do not need to know in advance where the data comes from or how it is obtained.) In this sense, the IOC can be regarded as a special type of server that handles process data and connects to real field hardware (in many cases, but not necessarily.)

The EPICS software components Channel Access (CA) and pvAccess (PVA) provide the protocols and structures that enable network transparent communication between client software running on a CWS and an arbitrary number of IOCs and other servers. More details about CA and PVA are provided in later chapters.

1.1.2 Basic Attributes

The basic attributes of EPICS are:

- Tool Based: EPICS provides a set of interacting tools and components for creating a control system. This minimizes the need for customer-specific coding and helps ensure uniform operator interfaces.
- Distributed: An arbitrary number of IOCs and CWSs can be supported. As long as the network is not saturated, there is not a single bottleneck. If a single IOC becomes saturated, its functions can be spread over several IOCs. Rather than running all applications on a single CWS host, the applications can be spread over many CWSs.
- Event Driven: The EPICS software components are all designed to be event driven to the maximum extent possible. For example, an EPICS client may, instead of having to query IOCs for changes, request to be notified of a change. This design leads to efficient use of resources, as well as quick response times.
- High Performance: An IOC can process tens of thousands of data items ("database records", see below) per second. Clients and servers can handle systems with millions of process variables, with minimized network overhead.
- Scalable: As a distributed system, EPICS can scale from systems with a single IOC and a few clients to large installations with hundreds of IOCs and millions of I/O channels and process variables.
- Robust: failure of a single components does not bring the whole system down. Components (IOCs, clients) can be added to and removed from the system without having to stop operation of the control system. The components can withstand intermittent failures of the interconnecting network and recover automatically when the network recovers from failure.

- Process-variable based: In contrast to some other control system packages, EPICS does not model control system (I/O) devices as objects (as in object-oriented programming) but rather as data entities that describe a single aspect of the process or device under control, thus the name "process variable", or "PV". A typical PV can represent any one of various attributes such as temperature or (electric) current. This design is typical in process control systems. The pros and cons of this design are shortly discussed in the Appendix.

1.1.3 IOC Software Components

An EPICS IOC at its core is a software entity or a process that contains the following software components:

- IOC Database: A memory resident database containing a set of named records of various types. The records host the process variables that were mentioned above.
- Scanners: The mechanisms for processing records in the IOC database.
- Record Support: Each record type has an associated set of record support routines to implement the functionality of the record type.
- Device Support: Device support routines bind I/O data to the database records.
- Device Drivers: Device drivers handle access to external devices.
- Channel Access or pvAccess: The interface between the external world and the IOC. It provides the interface for accessing the (EPICS) database via the network.
- Sequencer: A finite state machine. Strictly speaking, this is an external module and not included in the EPICS core software distribution.

Let us briefly describe the major components of the IOC and how they interact.

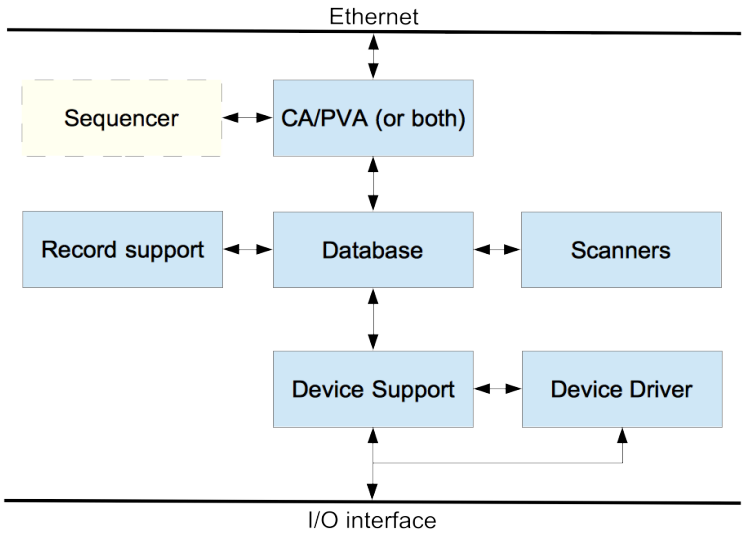


Figure 2. EPICS IOC components.

1.1.4 IOC Database

The heart of each IOC is a process database. This database is memory resident (i.e., not stored on a hard disk or other permanent memory device) and has nothing to do with the more commonly known relational (aka SQL) databases.

The database defines the functionality of the IOC: what process data it provides, how is the data handled and stored. The database can contain any number of records, each of which belongs to a specific record type. The record type defines the type of data that the record handles and a set of functions that define how the data are handled. Record type-specific metadata, also known as "properties" is included in the records to configure and support the operation. For instance, an analog input (ai) record type supports reading in values from hardware devices and converting them into desired (engineering) units. It also provides limits for expected operating ranges and alarms when these limits are exceeded. EPICS supports a large and extensible set of record types, e.g. ai (Analog Input), ao (Analog Output), etc.

The metadata, known as "fields" is used to configure the record's behavior. There are a number of fields that are common to all record types while some fields are specific to particular record types. Every record has a record name and every field has a field name. The record name must be unique across all IOCs that are attached to the same TCP/IP subnet, to enable the client software to discover any record on the subnet and to access its value and other fields.

```
record(ai, "Cavity1:T") #type = ai, name = "Cavity1:T"
{
  field(DESC, "Cavity Temperature") #description
  field(SCAN, "1 second") #record update rate
  field(DTYP, "XYZ ADC") #Device type
  field(INP, "#C1 S4") #input channel
  field(PREC, "1") #display precision
  field(LINR, "typeJdegC") #conversion spec
  field(EGU, "degrees C") #engineering units
  field(HOPR, "100") #highest value on GUI
  field(LOPR, "0") #lowest value on GUI
  field(HIGH, "65") #High alarm limit
  field(HSV, "MINOR") #Severity of "high" alarm
}
```

Figure 3. Example of an EPICS database record. Only a subset of fields is defined here.

Database records can be linked with each other. For example, records can retrieve input from other records, trigger other records to process, enable or disable records and so on.

By linking a combination of records together, the EPICS database becomes a programming tool. Using this, even very sophisticated functions can be achieved with the database. In addition, as this logic resides on the IOC, it is not dependent on any client software to work. By taking advantage of this, many client programs can be "thin" and just display or write the values in the database records. Figure 4 below illustrates a simple example of record linking: if the average temperature of the two sensors T1 and T2 is over 10 degrees, the chiller is switched on. This database contains four records: two analog inputs (ai), one binary output (bo) and one calculation (calc).

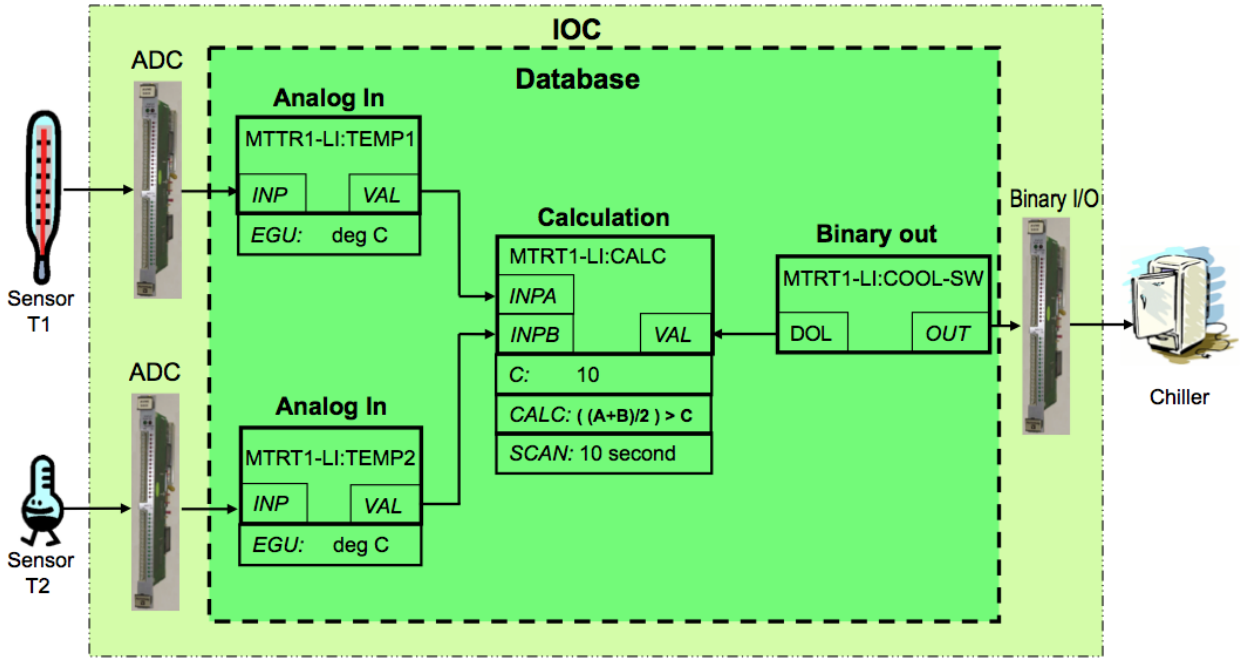


Figure 4. Example of record linking. From [2].

Data structures are provided so that the database can be accessed efficiently. Most software components do not need to be aware of these structures because they access the database via library routines.

1.1.5 Database Scanning

Database scanning is the mechanism to process a record. Processing means making the record perform its task, for instance reading an I/O channel, converting the read value to engineering units, attaching a timestamp to the value or checking the alarm limits. How data are handled when a record is processed depends on the record type.

Four basic types of record scanning are provided: Periodic, Event, I/O Event and Passive. All these methods can be mixed in an IOC.

- **Periodic**: A record is processed periodically. A number of time intervals are supported, typically ranging from 10 Hz to 0.01 Hz. Ranges are configurable to support higher and lower rates.
- **Event**: Event scanning happens when any IOC software component posts an (EPICS software) event, such as a new temperature sensor measurement value.
- **I/O Event**: The I/O event scanning system processes records based on external events like processor interrupts. An IOC device driver interrupt routine must be available to accept the external interrupts. An I/O Event does not necessarily have to be an interrupt in the traditional sense of a CPU interrupt, though.
- **Passive**: Passive records are not scanned regularly or on events. However, they can be processed as a result when other records that are linked to them are processed, or as a result of external changes such as new values set over network using Channel Access.

1.1.6 Record Support, Device Support and Device Drivers

Access to the database does not require record type-specific knowledge; each record type provides a set of record support routines that implement all record-specific behavior. Therefore, IOCs can support an arbitrary number of records and record types. Similarly, record support contains no device specific knowledge, giving each record type the ability to have any number of independent device support modules. If the method of accessing the piece of hardware is more complicated than can be handled by device support, then a device driver can be developed. Sometimes splitting functionality between device support (when it is record type-specific) and a driver (when the code handles device-specific details) is a good practice.

Record types that are not associated with hardware do not need to have device support or device drivers. One example is a calculation ("calc") record that reads its input from other records, performs a calculation and then (optionally) forwards the result to other records.

The IOC software design allows a particular installation and even a particular IOC within an installation to choose a unique set of record types, device types, and drivers. The remainder of the IOC system software is unaffected.

To give an overview of how the separation works, let us look at the tasks of the record support. Every record support module must provide a record processing routine to be called by the database scanners. Record processing consists of some combination of the following functions (all record types do not need all functions):

- **Input**: Read inputs. Inputs can be obtained, via device support routines, from hardware, from other database records via database links, or from other IOCs via Channel Access (CA) or pvAccess (PVA) links.
- **Conversion**: Conversion of raw input to engineering units or engineering units to raw output values.
- **Output**: Write outputs. Output can be directed, via device support routines, to hardware, to other database records within the same IOC via database links, or to other IOCs via CA or PVA links.
- **Raise Alarms**: Check for and raise alarms.
- **Monitor**: Trigger monitors related to CA or PVA callbacks.
- **Link**: Trigger processing of linked records.

The same concept is applied to the device support and device driver modules: each support module has to define a set of functions so that it can become a part of the IOC software.

1.1.7 Database Monitors

The mechanism to send notifications when a database value changes is called "database monitors". The monitor facility allows a client program to be notified when database values change without having to constantly poll the database. These can be configured to specify value changes, alarm changes, and/or archival changes.

Database monitors are supported by the EPICS standard protocols Channel Access and pvAccess.

1.1.8 Network protocols

EPICS provides network transparent access to IOC databases by supporting the following network protocols for data exchange.

Channel Access

Channel Access is based on a client/ server model. Each IOC provides a Channel Access server that is able to establish communication with an arbitrary number of clients. Channel Access client services are available on both CWSs and IOCs. A client can communicate with an arbitrary number of servers.

Client Services

The basic Channel Access client services are:

- **Search**: Locate the IOCs containing selected process variables and establish communication with each one.
- **Get**: Get value plus additional optional information for a selected set of process variables.
- **Put**: Change the values of selected process variables.
- **Monitor**: Request to have the server send information only when the associated process variable changes state. Any combination of the following state changes can be requested: change of value, change of alarm status and/or severity, and change of archival value. Many record types provide hysteresis factors for value changes.

In addition to process variable values, any combination of the following additional information ("metadata") may be requested:

- **Status**: Alarm status and severity.
- **Units**: Engineering units for this process variable.
- **Precision**: Precision with which to display floating-point numbers.
- **Timestamp**: Time when the record was last processed.
- **Enumeration**: A set of ASCII strings defining the meaning of enumerated values.
- **Graphics**: High and low limits for configuring widgets and graphs on a graphical user interface (GUI).
- **Control**: High and low control limits; operational limits for the record.
- **Alarm**: The alarm status (HIHI, HIGH, LOW, and LOLO) and severity for the process variable.

Search Server

Channel Access provides an IOC resident server, which waits for Channel Access search messages. These are UDP broadcasts that are generated by a Channel Access client (for example when an Operator Interface task starts) when it searches for the IOCs containing process variables it uses. This server accepts all search messages, checks to see if any of the process variables are located in this IOC, and, if any are found, replies to the sender with an "I have it" message.

Connection Request Server

Once the process variables have been located, the Channel Access client issues connection requests for each IOC containing process variables the client uses. The connection request server, in the IOC, accepts the request and establishes a connection to the client. Each connection is managed by two separate tasks: ca_get and ca_put. The ca_add_event requests result in database monitors being established. Database access and/or record support routines provide the value updates (monitors) via a call to db_post_event.

Connection Management

Each IOC provides a connection management service. If a Channel Access server fails (e.g. its IOC crashes) the client is notified and when a client fails (e.g. its task crashes) the server is notified. If a client fails, the server breaks the connection. If a server crashes, the client automatically re-establishes communication when the server restarts.

pvAccess

pvAccess is a modern replacement and an alternative to Channel Access available in EPICS 7. PvAccess adds a number of capabilities to EPICS that augment the set of services provided by Channel Access. With pvAccess, structured data can be transported with a high efficiency and is capable of handling big data sets; this has been achieved with a number of optimizations:

- Data structure introspection and data transport have been separated so that structure information needs to be carried only once per connection.
- Monitors send only the items of a data structure that have changed.
- Several under-the-hood optimizations in data manipulation have been made (reduce copying, etc.) In application testing pvAccess has been able to utilize 96-99% percent of the available theoretical bandwidth of a 10 Gbit Ethernet link which is close to the limit of what is achievable in practice.

Client Services

The basic pvAccess client services are similar to Channel Access, with a couple of additions:

- **Search**: Locate the IOCs that contain the process variables of interest and establish communication with each one.
- **Get**: Get value plus additional optional information for a selected set of process variables.
- **Put**: Change the values of selected process variables.
- **Add Monitor**: Add a change of state callback, similar to Channel Access.
- **PutGet**: Change the value of a PV, process the EPICS record and read back the value in one atomic operation.
- **ChannelRPC**: A "Remote Procedure Call" [3] communication pattern. This is similar to PutGet, but the communication is asymmetric, i.e., the data sent by client ("request") is different from the data structure that the server sends back. This pattern can be described as a query with parameters. Examples could be to ask a calibration service for parameters for a certain device, or a beam physics server for calculated beam parameters at certain coordinates of the accelerator.

For the IOC, an IOC resident server (**qsrv**) provides the interface to access the process database records. Basic access to a single PV provides the equivalent function to channel access. In addition, qsrv provides the possibilities to create data structures that combine data from different database records into structures that are transported as units. Since EPICS 3.16, the IOC core is able to guarantee atomic access to the records, meaning that the data in the structure that qsrv provides is guaranteed to be a result of a single processing (or better expressed, that the records do not change their values while qsrv is assembling the data structure.) This applies also to puts, meaning that all values are written to the addressed records before the records are processed. This way, coherence of parameters for an operation can be guaranteed.

Search Server

Like in Channel Access, **qsrv** waits for search messages. The server accepts all (UDP) search messages, checks to see if any of the process variables are located in this IOC, and if any are found, replies to the sender with an "I have it" message.

Connection Request Server

In pvAccess, the process of how a client and a server establish the communication channel is slightly different from Channel Access and contains two stages. The first stage is exchanging introspection data. In this stage, the server communicates to the client the structure of the data to be exchanged. Both sides can then create the necessary placeholder structures for the communication. In the second stage the actual data can be exchanged, using the allocated data structures.

Connection Management

pvAccess provides a connection management service similar to Channel Access.

EPICS database and network transport

It should be noted that the access methods (pvAccess, Channel Access) do **not** provide access to the EPICS database as records. This is a deliberate design decision. This allows changes to be made in the database structures or new record types to be added without impacting any software that accesses the database via PVA or CA, and it allows these clients to communicate with multiple IOCs having differing sets of record types.

1.1.9 Client Workstation Tools

EPICS offers a range of tools and services that are executed on the client workstations. These can be divided into two groups based on whether or not they use Channel Access and/or pvAccess. CA/PVA tools are real time tools, i.e. they are used to monitor and control IOCs. These tools are not included in the EPICS "base" distribution and have to be downloaded separately. The tools are implemented in different languages and technologies and the users should select which tools are the best suited to their particular setup and infrastructure.

Examples of CA/pvAccess Tools

A large number of CA/PVA tools have been developed. The following are some representative examples.

- CS-Studio: Control System Studio, an application bundle with many available plug-ins like display managers (BOY, Display Builder), data visualization/charting tools (DataBrowser), and so on.
- EDM: Extensible Display Manager. One of the several alternative display managers. Other popular alternatives are caQtDM (based on the Qt framework), medm (Motif Extended Display Manager, a legacy tool), just to name a couple.
- Alarm Handler. General-purpose alarm handler driven by an alarm configuration file.
- Sequencer: Runs in an IOC to implement state machines.
- Archiver Appliance: Collects data from EPICS servers (CA,PVA) and stores the data in time-series files so that they can be later retrieved and analyzed for correlating events and monitoring the performance of the "machine", i.e., the device or facility under control.
- Channel Finder (Indexing Service): A tool to manage (list, tag, categorize) the EPICS records in a system. This is a powerful tool to manage and provide hierarchy and different viewpoints to the potentially very large number of records. With this service, abstract views to the flat namespace of the records can be provided. For example,

listing all vacuum pumps in the system, or horizontal position of the beam in the accelerator as measured by the Beam Position Monitors.

Examples of other Tools

- VDCT: A Java based database configuration tool, which can be used to design and configure EPICS databases, and is able to visualize the records and their connections.
- SNC: State Notation Compiler. It generates a C program that represents the states for the IOC Sequencer tool.

1.1.10 References and further reading

1. Control Theory (https://en.wikipedia.org/wiki/Control_theory)
2. http://epics.web.psi.ch/training/handouts/e_EPICS_Training_at_PSI.ppt
3. https://en.wikipedia.org/wiki/Remote_procedure_call
4. EPICS Application Developer's Manual (version dependent, see for instance http://www.aps.anl.gov/epics/base/R3-15/5-docs/AppDevGuide/AppDevGuide.html)
5. https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/atomic-action
6. Recent Advancements and Deployments of EPICS Version 4, Greg White et. al., ICALEPCS 2015, Melbourne, Australia.

1.1.11 Appendix: Objects vs Process Variables discussion

As discussed in Chapter 2, EPICS is based on a "flat", i.e., non-hierarchical set of records, which represent the Process Variables[1] of the control system. This has a number of pros and cons:

Pros:

- Easy to adjust to any specific case without need of detailed modeling of the devices.
- Efficient communication: only the data of interest needs to be transported.
- PVs are modular building blocks that can be mixed and matched as needed.
- Even complex functionality can be implemented without (traditional) programming.

Cons:

- Lack of abstraction; control of complex entities has to be implemented on top of the PVs.
- Management of discrete data items is hard; lack of atomic actions [4].
- Advantages of object-oriented programming (code reuse, encapsulation, etc.) cannot be utilized.

One can extend these lists and argue about them but the above are the most common.

There is no single truth saying that this model is better or worse than other conceivable models. It depends on the use case and how much weight is put on each different factor.

However, the new features in EPICS 7 have been added to mitigate the lack of abstraction and atomic actions. The structured data model in EPICS 7 allows construction of complex structures to represent abstract entities. Further, these entities can be built from the existing building blocks, thus the flexibility is retained; in a way this is even better than strict modeling because the abstraction can be added on top of the working system afterwards. Also, atomic

[1] Strictly speaking, each field of a record can also be considered as a process variable. However, for this discussion it is sufficient to take the simpler approach to equate a record with a PV.

actions – to the extent they can be implemented in a distributed system – have been added, thus removing the need of complicated workaround solutions.

1.2 Installation Overview

Tags: beginner

An EPICS installation typically consists of multiple software modules.

EPICS Base will always be one of them. Base and additional modules that provide libraries or tools are often referred to as *Support Modules*, while the modules that produce your control system are often called *IOC Application Modules*.

EPICS Base and the Support Modules are usually common and shared between the IOC Applications of an installation. You can consider a stable and tested set of Base and Support Modules a *release* of your development environment.

As Support Modules are shared, have a longer life cycle and are held more stable than the IOC Applications that use them, it is a good idea to keep the Support Modules and IOC Applications separate.

This section will mostly cover installing EPICS Base and Support Modules. IOC Applications are too specific to be covered by general documentation.

1.2.1 General workflow

The traditional way to install EPICS is by compiling from sources.

While the specific instructions differ between Operating Systems on your host, the general steps are always the same:

1. Install prerequisites
2. Download, configure and install EPICS Base
3. Download, configure and install Support Modules
4. Create your IOC Application

1.2.2 Which version should I chose?

Please use new versions.

Unless you have specific reasons to use an older version, using the current release will make sure you have all the features and all the bug fixes. Using current versions for *all* modules in your set of Support Modules minimizes issues that may show up because of incompatibilities.

1.3 Installation on Linux / MacOS

1.3.1 Scope of these instructions

Starting from scratch, we get to the point where we have a working server, offering some PVs for reading (caget or pvget) and writing (caput or pvput). We read and write on them from another terminal, either on the same machine or on another one in the same network.

If you are using two different machines, both of them need to have access to the same EPICS installation.

1.3.2 Prepare your system

You need `make`, `c++` and `libreadline` to compile from source. On macOS these dependencies can be installed by using e.g. `homebrew`. On Linux you can use `apt-get install`. The *EPICS Dependencies on CentOS 8* document lists all of the packages required to build EPICS base, the sequencer, synApps modules, and areaDetector.

1.3.3 Install EPICS

```
mkdir $HOME/EPICS
cd $HOME/EPICS
git clone --recursive https://github.com/epics-base/epics-base.git
cd epics-base
make
```

After compiling you should put the path into `$HOME/.profile` or into `$HOME/.bashrc` by adding the following to either one of those files:

```
export EPICS_BASE=${HOME}/EPICS/epics-base
export EPICS_HOST_ARCH=$(${EPICS_BASE}/startup/EpicsHostArch)
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:${PATH}
```

EpicsHostArch is a program provided by EPICS that returns the architecture of your system. Thus the code above should be fine for every architecture.

1.3.4 Test EPICS

Now log out and log in again, so that your new path is set correctly. Alternatively, you can execute the three lines above beginning with export directly from the terminal.

Run `softIoc` and, if everything is ok, you should see an EPICS prompt.

```
softIoc
epics>
```

You can exit with ctrl-c or by typing exit.

Voilà.

Ok, that is not very impressive, but at least you know that EPICS is installed correctly. So now let us try something more complex, which will hopefully suggest how EPICS works.

In whatever directory you like, prepare a file test.db that reads like

```
record(ai, "temperature:water")
{
    field(DESC, "Water temperature in the fish tank")
}
```

This file defines a record instance called temperature:water, which is an analog input (ai) record. As you can imagine DESC stays for Description. Now we start softIoc again, but this time using this record database.

```
softIoc -d test.db
```

Now, from your EPICS prompt, you can list the available records with the dbl command and you will see something like

```
epics> dbl
temperature:water
epics>
```

Open a new terminal (we call it nr. 2) and try the command line tools caget and caput. You will see something like

```
your prompt> caget temperature:water
temperature:water              0
your prompt> caget temperature:water.DESC
temperature:water.DESC         Water temperature in the fish tank
your prompt> caput temperature:water 21
Old : temperature:water              0
New : temperature:water              21
your prompt> caput temperature:water 24
Old : temperature:water              21
New : temperature:water              24
your prompt> caget temperature:water
temperature:water              24
 ... etc.
```

Now open yet another terminal (nr. 3) and try `camonitor` as

```
camonitor temperature:water
```

First, have a look at what happens when you change the temperature:water value from terminal nr. 2 using caput. Then, try to change the value by some tiny amounts, like 15.500001, 15.500002, 15.500003... You will see that camonitor reacts but the readings do not change. If you wanted to see more digits, you could run

```
camonitor -g8 temperature:water
```

For further details on the Channel Access protocol, including documentation on the `caput`, `caget`, `camonitor`... command line tools, please refer to the Channel Access Reference Manual.

In real life, however, it is unlikely that the 8 digits returned by your thermometer (in this example) are all significant. We should thus limit the traffic to changes of the order of, say, a hundredth of a degree. To do this, we add one line to the file `test.db`, so that it reads

```
record(ai, "temperature:water")
{
    field(DESC, "Water temperature in Lab 10")
    field(MDEL, ".01")
}
```

MDEL stands for Monitor Deadband. If you now run

```
softIoc -d test.db
```

with the new `test.db` file, you will see that `camonitor` reacts only to changes that are larger than 0.01.

This was just a simple example. Please refer to a recent Record Reference Manual for further information.

1.3.5 Create a demo/test ioc to test ca and pva

```
mkdir -p $HOME/EPICS/TEST/testIoc
cd $HOME/EPICS/TEST/testIoc
makeBaseApp.pl -t example testIoc
makeBaseApp.pl -i -t example testIoc
make
cd iocBoot/ioctestIoc
chmod u+x st.cmd
ioctestIoc> ./st.cmd
#!../../bin/darwin-x86/testIoc
< envPaths
epicsEnvSet("IOC","ioctestIoc")
epicsEnvSet("TOP","/Users/maradona/EPICS/TEST/testIoc")
epicsEnvSet("EPICS_BASE","/Users/maradona/EPICS/epics-base")
cd "/Users/maradona/EPICS/TEST/testIoc"
## Register all support components
dbLoadDatabase "dbd/testIoc.dbd"
testIoc_registerRecordDeviceDriver pdbbase
## Load record instances dbLoadTemplate "db/user.substitutions"
dbLoadRecords "db/testIocVersion.db", "user=junkes"
dbLoadRecords "db/dbSubExample.db", "user=junkes"
#var mySubDebug 1
#traceIocInit
cd "/Users/maradona/EPICS/TEST/testIoc/iocBoot/ioctestIoc"
iocInit
Starting iocInit
############################################################################
## EPICS R7.0.1.2-DEV
## EPICS Base built Mar 8 2018
############################################################################
iocRun: All initialization complete
2018-03-09T13:07:02.475 Using dynamically assigned TCP port 52908.
## Start any sequence programs
#seq sncExample, "user=maradona"
epics> dbl
maradona:circle:tick
maradona:compressExample
maradona:line:b
maradona:aiExample
maradona:aiExample1
maradona:ai1
maradona:aiExample2
... etc. ...
epics>
```

Now in another terminal, one can try command line tools like

```
caget, caput, camonitor, cainfo (Channel Access)
pvget, pvput, pvlist, eget, ... (PVAccess)
```

1.3.6 Add the asyn package

```
cd $HOME/EPICS
mkdir support
cd support
git clone https://github.com/epics-modules/asyn.git
cd asyn
```

Edit `$HOME/EPICS/support/asyn/configure/RELEASE` and set `EPICS_BASE` like

```
EPICS_BASE=${HOME}/EPICS/epics-base
```

Comment `IPAC=...` and `SNCSEQ=...`, as they are not needed for the moment. The whole file should read:

```
#RELEASE Location of external products
HOME=/Users/maradona
SUPPORT=$(HOME)/EPICS/support
-include $(TOP)/../configure/SUPPORT.$(EPICS_HOST_ARCH)
# IPAC is only necessary if support for Greensprings IP488 is required
# IPAC release V2-7 or later is required.
#IPAC=$(SUPPORT)/ipac-2-14
# SEQ is required for testIPServer
#SNCSEQ=$(SUPPORT)/seq-2-2-5
# EPICS_BASE 3.14.6 or later is required
EPICS_BASE=$(HOME)/EPICS/epics-base
-include $(TOP)/../configure/EPICS_BASE.$(EPICS_HOST_ARCH)
```

Now, run

```
make
```

If the build fails due to implicit declaration of `xdr_*` functions it is likely that asyn should build against libtirpc. To do so, you can uncomment `# TIRPC=YES` in `configure/CONFIG_SITE` of asyn, such that it states:

```
# Some linux systems moved RPC related symbols to libtirpc
# To enable linking against this library, uncomment the following line
TIRPC=YES
```

1.3.7 Install StreamDevice (by Dirk Zimoch, PSI)

```
cd $HOME/EPICS/support
git clone https://github.com/paulscherrerinstitute/StreamDevice.git
cd StreamDevice/
rm GNUmakefile
```

Edit `$HOME/EPICS/support/StreamDevice/configure/RELEASE` to specify the install location of EPICS base and of additional software modules, for example:

```
EPICS_BASE=${HOME}/EPICS/epics-base
SUPPORT=${HOME}/EPICS/support
ASYN=$(SUPPORT)/asyn
```

Remember that `$(NAME)` works if it is defined within the same file, but `${NAME}` with curly brackets must be used if a shell variable is meant. It is possible that the compiler does not like some of the substitutions. In that case, replace the `${NAME}` variables with full paths, like `/Users/maradona/EPICS...`.

The sCalcout record is part of synApps. If streamDevice should be built with support for this record, you have to install at least the calc module from SynApps first. For now let's just comment out that line with # for it to be ignored.

::

    #CALC=${HOME}/EPICS/support/synApps/calc

If you want to enable regular expression matching, you need the PCRE package. For most Linux systems, it is already installed. In that case tell StreamDevice the locations of the PCRE header file and library. However, the pre-installed package can only by used for the host architecture. Thus, add them not to RELEASE but to RELEASE.Common.linux-x86 (if linux-x86 is your EPICS_HOST_ARCH). Be aware that different Linux distributions may locate the files in different directories.

```
PCRE_INCLUDE=/usr/include/pcre
PCRE_LIB=/usr/lib
```

For 64 bit installations, the path to the library may be different:

```
PCRE_INCLUDE=/usr/include/pcre
PCRE_LIB=/usr/lib64
```

Again, if you're not interested in support for reular expression matching at this time then you can comment out any lines referring to PCRE in the `configure/RELEASE` file using a #. It can always be added later.

Finally run `make` (we are in the directory `...EPICS/support/StreamDevice`)

1.4 Installation on Windows

Tags: beginner

1.4.1 Introduction

EPICS

EPICS is a toolkit for building control systems. You can get the basic ideas from the EPICS web site at https://epics-controls.org/about-epics/.

Traditionally, an EPICS installation starts with compiling the core parts ("EPICS Base") from source. This process is covered by these instructions, starting from scratch on a Windows system and getting you to the point where you have a working IOC and can connect to it from a command line shell. Other How-Tos will guide you further.

EPICS on Windows

While it is not its primary or most widely used target platform, the EPICS low-level libraries have good and well-tested implementations on Windows. EPICS runs fine on Windows targets, fast and robust.

There are, however, a few choices about how to compile and run EPICS on Windows that you will have to take beforehand. Understanding these choices and their implications before making decisions will help you to avoid mistakes and spend time fixing them.

Cygwin

As mentioned before, EPICS Base has its own native Windows implementation of all necessary low level services. There is no need to go through the Posix emulation layer that Cygwin provides. The native Windows implementation is more portable and performs better. Unless you need to use Cygwin, e.g., if you are using a binary vendor-provided library for Cygwin, you should prefer a native Windows build.

Also, Cygwin is deprecated as a target platform for EPICS.

Build Time

The time needed to build EPICS Base depends on a few factors, including the speed of the processor and file system, the compiler used, the build mode (DLL or static), possibly debugging options and others. On a medium sized two-core machine, a complete build of EPICS 7 often takes between 15 and 30 minutes, the 3.15 branch can be built in 6 to 10 minutes.

Use `make -j<n>` to make use of multiple CPU cores.

1.4.2 Required Tools

- C++ compiler: either MinGW (GCC) or Microsoft's Visual Studio compiler (VS)
- archive unpacker (7zip or similar)
- GNU Make (4.x)
- Perl

1.4.3 Choice 1: Compiler

You will need a C++ compiler with its supporting C++ standard libraries. Two major compilers are supported by EPICS and its build system:

Microsoft's Visual Studio compiler (VS)
: Probably the most widely used compiler for EPICS on the Windows platform. The "Community Edition" is free to download and use. (You need to have Administrator rights to install it.) Any Visual Studio installation will need the "C++ development" parts for the compiler toolchain to be installed.

  EPICS is using the Make build system. You can use the Visual Studio IDE, but EPICS does not provide any project files or configurations for Visual Studio's own build system.

MinGW (GCC) - Minimalist GNU for Windows
: A compiler toolchain based on the widely-used GNU compilers that - like the VS compiler - generates native Windows executables.

Both compiler toolchains can create shared libraries (DLLs) and static libraries. On a 64bit system, both can create 64bit output (runs on 64bit systems) and 32bit output (runs on both 32bit and 64bit systems).

When using C++, libraries are not compatible between those two compilers toolchains. When generating a binary (e.g., an IOC), all C++ code that is being linked must have been generated uniformly by either VS or MinGW. (The reason is different name mangling for symbol names: a symbol needed for linking an executable will not be found in a library generated with the other compiler, because its name is different there.)

If you need to link against vendor-provided binary C++ libraries, this will most likely determine which compiler you need to use.

1.4.4 Choice 2: Build Environment and Tool Installation

MSYS2

MSYS2 (available for Windows 7 and up) is a pretty complete "feels like Linux" environment. It includes a Linux style package manager (*pacman*), which makes it very easy to install the MinGW toolchains (32 and 64 bit) and all other necessary tools. It also offers a bash shell. If you are used to working in a Linux environment, you will like working on MSYS2.

MSYS2 can be installed, used and updated (including tools and compilers) without Administrator rights.

As up-to-date MinGW/GCC compilers are an integral part of the package, MSYS2 is strongly recommended for using the MinGW compiler toolchains.

The Visual Studio compilers can also be used from the MSYS2 bash. This needs a one-time setup of an intermediate batch script to get the Visual Studio environment settings correctly inherited. The resulting shell can compile using Visual Studio compilers as well as using MinGW, selected by the EPICS_HOST_ARCH environment variable setting.

Chocolatey

Chocolatey is a package manager for Windows with a comfortable GUI, making it easy to install and update software packages (including the tools needed for building EPICS). In many cases, Chocolatey packages wrap around the native Windows installers of software.

Using Chocolatey needs Administrator rights.

Windows Installers

You can also install the required tools independently, directly using their native Windows installers.

For Perl, both Strawberry Perl and ActivePerl are known to work. Strawberry Perl is more popular; it includes GNU Make (as *gmake.exe*) and the MinGW/GCC compiler necessary to build the Channel Access Perl module that is part of EPICS Base.

For GNU Make, the easiest way is to use the one included in Strawberry Perl. Otherwise, there is a Windows binary provided on the EPICS web site.

Native Windows installers often need Administrator rights.

1.4.5 Choice 3: Static or DLL Build / Deployment

If you configure the EPICS build system to build your IOCs dynamically (i.e., using DLLs), they need the DLLs they have been linked against to be present on the target system, either in the same directory as the IOC binary or in a directory that is mentioned in the `%PATH%` environment variable.

Depending on how you plan to deploy your IOCs into the production system, it might be easier to use static builds when generating IOCs. The resulting binaries will be considerably larger, but they will run on any Windows system without providing additional EPICS DLLs.

When running many EPICS IOCs on a single target machine, the *shared* aspect of a DLL build will lead to smaller memory usage. The DLL is in memory once and used concurrently by all IOC binaries, while the statically linked binaries each have their own copy of the library in memory.

Note: When using the Visual Studio compilers, compilation uses different flags for building DLLs and building static libraries. You can't generate static and shared libraries in the same build. You can provide both options in your EPICS installation by running both builds in sequence (with `make clean` inbetween), so that your applications can decide between static or DLL build. Or you can just provide one option globally for your installation, which all installations will have to use.

1.4.6 Windows Path Names

Make based builds do not work properly when there are space characters or parentheses in the paths that are part of the build (including the path where the *make* application resides and the path of the workspace).

If you cannot avoid paths with such characters, use the Windows short path (can be displayed with `dir /x`) for all path components with those characters in any path settings and/or your workspace directory.

1.4.7 Put Tools in the PATH

No matter which shell and environment you use, the tools (perl, make) should end up being in the `%PATH%`, so that they are found when called just by their name.

1.4.8 Install and Build

Depending on your set of choices, the instructions for building EPICS Base, building IOC applications and running them are different. The following detailed instructions focus on two common sets of choices: using MSYS2 with the MinGW Gnu compilers and using the plain Windows command prompt with the Visual Studio compilers.

Setting the environment for building and running applications has to be done for either set of choices.

Installation using MSYS2 and the MinGW Compilers

MSYS2 has all the required tools available through an easy-to-use package manager, and its bash shell looks and feels like working on Linux. Most Bash commands are similar to their Linux versions. MSYS2 is available for Windows 7 and up only. The following procedure is verified on Windows 8.1 (64 bit) and Windows 10 (64 bit).

Install tools

MSYS2 provides a Bash shell, Autotools, revision control systems and other tools for building native Windows applications using MinGW-w64 toolchains. It can be installed from its official website. Download and run the installer - "x86_64" for 64-bit, "i686" for 32-bit Windows. The installation procedure is well explained on the website. These instructions assume you are running on 64-bit Windows.

The default location of the MSYS2 installation is `C:\msys64`. If you don't have Administrator rights, you can install MSYS2 in any location you have access to, e.g. `C:\Users\'user'\msys64` (with 'user' being your Windows user directory name). We will assume the default location in this document.

Once installation is complete, you have three options available for starting a shell. The difference between these options is the preset of the environment variables that select the compiler toolchain to use. Launch the "MSYS MinGW 64-bit" option to use the MinGW 64bit toolchain, producing 64bit binaries that run on 64bit Windows systems. The "MSYS MinGW 32-bit" option will use the MinGW 32bit toolchain, producing 32bit binaries that will run on 32bit and 64bit Windows systems.

Again: you have a single installation of MSYS2, these different shells are just setups to use different compilers. Installation and update of your MSYS2 system only has to be done once - you can use any of the shell options for that.

Update MSYS2 with following command:

```
$ pacman -Syu
```

After this finishes (let it close the bash shell), open bash again and run the same command again to finish the updates. The same procedure is used for regular updates of the MSYS2 installation. An up-to-date system shows:

```
$ pacman -Syu
:: Synchronizing package databases...
 mingw32 is up to date
 mingw64 is up to date
 msys is up to date
:: Starting core system upgrade...
 there is nothing to do
:: Starting full system upgrade...
 there is nothing to do
```

Install the necessary tools (perl is already part of the base system):

```
$ pacman -S tar make
```

Packages with such "simple" names are part of the MSYS2 environment and work for all compilers/toolchains that you may install on top on MSYS2.

Install compiler toolchains

Packages that are part of a MinGW toolchain start with the prefix "mingw-w64-x86_64-" for the 64bit toolchain or "mingw-w64-i686-" for the 32bit toolchain. (The "w64" part identifies the host system will be different when using a 32bit MSYS2 environment, e.g. on a 32bit Windows host.)

Install the MinGW 32bit and/or MinGW 64bit toolchains:

```
$ pacman -S mingw-w64-x86_64-toolchain
$ pacman -S mingw-w64-i686-toolchain
```

Each complete toolchain needs about 900MB of disk space. If you want to cut down the needed disk space (to about 50%), instead of hitting return when asked which packages of the group to install, you can select the minimal set of packages required for compiling EPICS Base: `binutils`, `gcc` and `gcc-libs`.

If you are not sure, check your set of tools is complete and everything is installed properly:

```
$ pacman -Q
...
make 4.3-1
perl 5.32.0-2
mingw-w64-x86_64-...
mingw-w64-i686-...
...
```

Update your installation regularly

As mentioned above, you can update your complete installation (including all tools and compiler toolchains) at any time using:

```
$ pacman -Syu
```

You should do this in regular intervals.

Download and build EPICS Base

Start the "MSYS MinGW 64-bit" shell and do:

```
$ cd $HOME
$ wget https://epics-controls.org/download/base/base-7.0.4.1.tar.gz
$ tar -xvf base-7.0.4.1.tar.gz
$ cd base-R7.0.4.1
$ export EPICS_HOST_ARCH=windows-x64-mingw
$ make
```

When using the MinGW 32bit toolchain, the "MSYS MinGW 32-bit" shell must be used and EPICS_HOST ARCH must be set to "win32-x86-mingw".

Note: If you are connecting to your MSYS2 system through ssh, you need to set and allow an environment variable to use the environment presets for the MinGW compilers. In the MSYS2 configuration of the ssh daemon (`/etc/ssh/sshd_config`), add the line

```
AcceptEnv MSYSTEM
```

and on your (local) client configuration (`~/.ssh/config`) add the line

```
SetEnv MSYSTEM=MINGW64
```

to use the MinGW 64-bit compiler chain (`MINGW32` to use a 32-bit installation).

During the compilation, there will probably be warnings, but there should be no error. You can choose any EPICS Base version to build, the procedure remains the same.

Please refer to the chapter "Build Time" in *Installation on Windows* for ways to shorten your build.

Quick test from MSYS2 Bash

As long as you haven't added the location of your programs to the `PATH` environment variable (see below), you will have to provide the whole path to run commands or *cd* into the directory they are located in and prefix the command with `./`.

Replace 'user' with the actual Windows user folder name existing in your Windows installation - MSYS2 creates your home directory using that name. In the examples, we assume the default location for MSYS2 (`C:\msys64`).

Run `softIoc` and, if everything is ok, you should see an EPICS prompt:

```
$ cd /home/'user'/base-R7.0.4.1/bin/windows-x64-mingw
$ ./softIoc -x test
Starting iocInit
iocRun: All initialization complete
dbLoadDatabase("C:\msys64\home\'user'\base-R7.0.4.1\bin\windows-x64-mingw\..\..\dbd\
↪softIoc.dbd")
softIoc_registerRecordDeviceDriver(pdbbase)
iocInit()
############################################################################
## EPICS R7.0.4.1
## Rev. 2020-10-21T11:57+0200
############################################################################
epics>
```

You can exit with ctrl-c or by typing exit.

As long as you are in the location of the EPICS Base binaries, you can run them by prefixing with `./`. Try commands like `./caput`, `./caget`, `./camonitor`, ...

Quick test from Windows command prompt

Open the Windows command prompt. Again, 'user' is the Windows user folder name. The MSYS2 home folders are inside the MSYS2 installation.

If you built EPICS Base with dynamic (DLL) linking, you need to add the location of the C++ libraries to the *PATH* variable for them to be found. (Again, assuming a 64bit MSYS2 installation with default paths and the MinGW 64bit toolchain.)

```
>set "PATH=%PATH%C:\msys64\mingw64\bin;"
>cd C:\msys64\home\'user'\base-R7.0.4.1\bin\windows-x64-mingw
>softIoc -x test
Starting iocInit
############################################################################
## EPICS R7.0.4.1
## Rev. 2020-10-21T11:57+0200
############################################################################
iocRun: All initialization complete
epics>
```

You can exit with ctrl-c or by typing exit.

As long as you are in the location of the EPICS Base binaries, they will all work using their simple names. Try commands like `caput`, `caget`, `camonitor`, ...

Create a demo/test IOC

Although the `softIoc` binary can be used with multiple instances with different db files, you will need to create your own IOC at some point. We will create a test ioc from the existing application template in Base using the `makeBaseApp.pl` script.

Let's create one IOC, which takes the values of 2 process variables (PVs), adds them and stores the result in 3rd PV.

We will use `MSYS2` for building the IOC. Open the `MSYS2 Mingw 64-bit` shell. Make sure the environment is set up correctly (see *Setting the system environment*).

Create a new directory `testioc`:

```
$ mkdir testioc
$ cd testioc
```

From that `testioc` folder run the following:

```
$ makeBaseApp.pl -t ioc test
$ makeBaseApp.pl -i -t ioc test
Using target architecture windows-x64-mingw (only one available)
The following applications are available:
    test
What application should the IOC(s) boot?
The default uses the IOC's name, even if not listed above.
Application name?
```

Accept the default name and press enter. That should generate a skeleton for your `testioc`.

You can find the full details of the application structure in the "Application Developer's Guide", chapter Example IOC Application.

```
$ ls
configure  iocBoot  Makefile  testApp
```

Now create a `db` file which describes PVs for your `IOC`. Go to `testApp/Db` and create `test.db` file with following record details:

```
record(ai, "test:pv1")
{
    field(VAL, 49)
}
record(ai, "test:pv2")
{
    field(VAL, 51)
}
record(calc,"test:add")
{
    field(SCAN, "1 second")
    field(INPA, "test:pv1")
    field(INPB, "test:pv2")
    field(CALC, "A + B")
}
```

Open `Makefile` and navigate to

```
#DB += xxx.db
```

Remove # and change this to `test.db`:

```
DB += test.db
```

Go to back to root folder for IOC `testioc`. Go to `iocBoot/ioctest`. Modify the `st.cmd` startup command file.

Change:

```
#dbLoadRecords("db/xxx.db","user=XXX")
```

to:

```
dbLoadRecords("db/test.db","user=XXX")
```

Save all the files and go back to the MSYS2 Bash terminal. Make sure the architecture is set correctly:

```
$ echo $EPICS_HOST_ARCH
windows-x64-mingw
```

Change into the testioc folder and run `make`:

```
$ cd ~/testioc
$ make
```

This should create all the files for the test IOC.

```
$ ls
bin  configure  db  dbd  iocBoot  lib  Makefile  testApp
```

Go to `iocBoot/ioctest` . Open the `envPaths` file and change the MSYS2 relative paths to full Windows paths:

```
epicsEnvSet("IOC","ioctest")
epicsEnvSet("TOP","C:/msys64/home/'user'/testioc")
epicsEnvSet("EPICS_BASE","C:/msys64/home/'user'/base-7.0.4.1")
```

Note: You can use Linux style forward slash characters in path specifications inside this file or double backslashes (\\).

At this point, you can run the IOC from either an MSYS2 Bash shell or from a Windows command prompt, by changing into the IOC directory and running the test.exe binary with your startup command script as parameter.

In the Windows command prompt:

```
>cd C:\msys64\home\'user'\testioc\iocBoot\ioctest
>..\..\bin\windows-x64-mingw\test st.cmd
```

In the MSYS2 shell:

```
$ cd ~/testioc/iocBoot/ioctest
$ ../../bin/windows-x64-mingw/test st.cmd
```

In both cases, the IOC should start like this:

```
Starting iocInit
iocRun: All initialization complete
#!../../bin/windows-x64-mingw/test
< envPaths
epicsEnvSet("IOC","ioctest")
epicsEnvSet("TOP","C:/msys64/home/'user'/testioc")
epicsEnvSet("EPICS_BASE","C:/msys64/home/'user'/base-R7.0.4.1")
cd "C:/msys64/home/'user'/testioc"
## Register all support components
dbLoadDatabase "dbd/test.dbd"
test_registerRecordDeviceDriver pdbbase
Warning: IOC is booting with TOP = "C:/msys64/home/'user'/testioc"
         but was built with TOP = "/home/'user'/testioc"
## Load record instances
dbLoadRecords("db/test.db","user='user'")
cd "C:/msys64/home/'user'/testioc/iocBoot/ioctest"
iocInit
############################################################################
## EPICS R7.0.4.1
## Rev. 2020-10-21T11:57+0200
############################################################################
## Start any sequence programs
#seq sncxxx,"user='user'"
epics>
```

Check if the database `test.db` you created is loaded correctly:

```
epics> dbl
test:pv1
test:pv2
test:add
```

As you can see 3 process variable is loaded and available. Keep this terminal open and running. Test this process variable using another terminals.

Open another shell for monitoring `test:add`:

```
$ camonitor test:add
test:add                       2020-10-23 13:39:14.795006 100
```

That terminal will monitor the PV `test:add` continuously. If any value change is detected, it will be updated in this terminal. Keep it open to observe the behaviour.

Open a third shell. Using caput, modify the values of `test:pv1` and `test:pv2` as we have done in the temperature example above. You will see changes of their sum in the second terminal accordingly.

At this point, you have one IOC `testioc` running, which loaded the database `test.db` with 3 records. From other processes, you can connect to these records using Channel Access. If you add more process variable in `test.db`, you will have to `make` the *testioc* application again and restart the IOC to load the new version of the database.

You can also create and run IOCs like this in parallel with their own databases and process variables. Just keep in mind that each record instance has to have a unique name for Channel Access to work properly.

Installation using plain Windows and the Visual Studio compilers

Install tools

There are two reasonable options.

Using Chocolatey

Go to the Chocolatey website and follow their instructions to download and install the package manager.

Using Chocolatey, install Strawberry Perl and Gnu Make.

Manually

Install Strawberry Perl or ActivePerl using the Windows installers available on their download pages.

Strawberry Perl contains a suitable version of GNU Make. Otherwise, you can download a Windows executable that Andrew provides at https://epics.anl.gov/download/tools/make-4.2.1-win64.zip. Unzip it into a location (path must not contain spaces or parentheses) and add it to the system environment.

Put tools in the Path

Make sure the tools' locations are added to the system environment variable Path. Inside a shell (command prompt) they must be callable using their simple name, e.g.:

```
>perl --version

This is perl 5, version 26, subversion 1 (v5.26.1) built for MSWin32-x64-multi-thread
(with 1 registered patch, see perl -V for more detail)

Copyright 1987-2017, Larry Wall

Binary build 2601 [404865] provided by ActiveState http://www.ActiveState.com
Built Dec 11 2017 12:23:25
...

>make --version
GNU Make 4.2.1
Built for x86_64-w64-mingw32
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Install the compiler

Download the Visual Studio Installer and install (the community edition is free). Make sure you enable the Programming Languages / C++ Development options.

In VS 2019, you also have the option to additionally install the Visual C++ 2017 compilers, if that is interesting for you.

Download and build EPICS Base

1. Download the distribution from e.g. https://epics-controls.org/download/base/base-7.0.4.1.tar.gz.
2. Unpack it into a work directory.
3. Open a Windows command prompt and change into the directory you unpacked EPICS Base into.

   Note: The complete path of the current directory mustn't contain any spaces or parentheses. If your working directory path does, you can do another cd into the same directory, replacing every path component containing spaces or parentheses with its Windows short path (that can be displayed with `dir /x`).
4. Set the EPICS host architecture EPICS_HOST_ARCH (windows-x64 for 64bit builds, win32-x86 for 32bit builds).
5. Run the `vcvarsall.bat` script of your installation (the exact path depends on the type and language of installation) to set the environment for your build.
6. Run `make` (or `gmake` if using the version from Strawberry Perl).

```
>cd base-R7.0.4.1
>set EPICS_HOST_ARCH=windows-x64
>"C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build\
↪vcvarsall.bat" amd64
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.6.2
** Copyright (c) 2020 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x64'

>make
```

There will probably be warnings, but there should be no error. You can choose any EPICS base to install, the procedure remains the same.

Please refer to the chapter "Build Time" in *Installation on Windows* for ways to shorten your build.

Quick test from Windows command prompt

As long as you haven't added the location of your programs to the `PATH` environment variable (see *Setting the system environment*), you will have to provide the whole path to run commands or *cd* into the directory they are located in.

Open the Windows command prompt. Again, replace 'user' with the actual Windows user folder name existing in your Windows installation.

Run `softIoc` and, if everything is ok, you should see an EPICS prompt:

```
>cd C:\Users\'user'\base-R7.0.4.1\bin\windows-x64-mingw
>softIoc -x test
Starting iocInit
iocRun: All initialization complete
dbLoadDatabase("C:\Users\'user'\base-R7.0.4.1\bin\windows-x64\..\..\dbd\softIoc.dbd")
softIoc_registerRecordDeviceDriver(pdbbase)
iocInit()
###############################################################################
## EPICS R7.0.4.1
## Rev. 2020-10-21T11:57+0200
###############################################################################
epics>
```

You can exit with ctrl-c or by typing exit.

As long as you are in the location of the EPICS Base binaries, they will all work using their simple names. Try commands like `caput`, `caget`, `camonitor`, ...

Quick test from MSYS2 Bash

Obviously, if you have an installation of MSYS2, you can run the same verification from the MSYS2 Bash shell:

```
$ cd /c/Users/'user'/base-R7.0.4.1/bin/windows-x64
$ ./softIoc -x test
Starting iocInit
iocRun: All initialization complete
dbLoadDatabase("C:\Users\'user'\base-R7.0.4.1\bin\windows-x64\..\..\dbd\softIoc.dbd")
softIoc_registerRecordDeviceDriver(pdbbase)
iocInit()
###############################################################################
## EPICS R7.0.4.1
## Rev. 2020-10-21T11:57+0200
###############################################################################
epics>
```

You can exit with ctrl-c or by typing exit.

As long as you are in the location of the EPICS Base binaries, you can run them by prefixing `./`. Try commands like `./caput`, `./caget`, `./camonitor`, ...

Create a demo/test IOC

Although the `softIoc` binary can be used with multiple instances with different db files, you will need to create your own IOC at some point. We will create a test ioc from the existing application template in Base using the `makeBaseApp.pl` script.

Let's create one IOC, which takes the values of 2 process variables (PVs), adds them and stores the result in 3rd PV.

We will use the Windows command prompt for building the IOC. Open the command prompt. Create a new directory `testioc`:

```
>mkdir testioc
>cd testioc
```

From that testioc folder run the following:

```
>makeBaseApp.pl -t ioc test
>makeBaseApp.pl -i -t ioc test
Using target architecture windows-x64 (only one available)
The following applications are available:
    test
What application should the IOC(s) boot?
The default uses the IOC's name, even if not listed above.
Application name?
```

Accept the default name and press enter. That should generate a skeleton for your testioc.

You can find the full details of the application structure in the "Application Developer's Guide", chapter Example IOC Application.

```
>dir /b
configure
iocBoot
Makefile
testApp
```

Now create a db file which describes PVs for your IOC. Go to testApp\Db and create test.db file with following record details:

```
record(ai, "test:pv1")
{
    field(VAL, 49)
}
record(ai, "test:pv2")
{
    field(VAL, 51)
}
record(calc,"test:add")
{
    field(SCAN, "1 second")
    field(INPA, "test:pv1")
    field(INPB, "test:pv2")
    field(CALC, "A + B")
}
```

Open Makefile and navigate to

```
#DB += xxx.db
```

Remove # and change this to test.db:

```
DB += test.db
```

Go to back to root folder for IOC testioc. Go to iocBoot\ioctest. Modify the st.cmd startup command file.

Change:

```
#dbLoadRecords("db/xxx.db","user=XXX")
```

to:

```
dbLoadRecords("db/test.db","user=XXX")
```

Save all the files and go back to the MSYS2 Bash terminal. Make sure the environment is set up correctly (see *Setting the system environment*).:

```
>echo $EPICS_HOST_ARCH
windows-x64
>cl
Microsoft (R) C/C++ Optimizing Compiler Version 19.27.29112 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.
```

Change into the testioc folder and run `make` (or `gmake` when using the make from Strawberry Perl):

```
>cd %HOMEPATH%\testioc
>make
```

This should build the executable and create all files for the test IOC:

```
>dir /b
bin
configure
db
dbd
iocBoot
lib
Makefile
testApp
```

At this point, you can run the IOC from either an MSYS2 Bash shell or from a Windows command prompt, by changing into the IOC directory and running the test.exe binary with your startup command script as parameter.

In the Windows command prompt:

```
>cd %HOMEPATH%\testioc\iocBoot\ioctest
>..\..\bin\windows-x64\test st.cmd
```

Or - if you have an installation - in the MSYS2 shell:

```
$ cd ~/testioc/iocBoot/ioctest
$ ../../bin/windows-x64/test st.cmd
```

In both cases, the IOC should start like this:

```
Starting iocInit
#!../../bin/windows-x64/test
< envPaths
epicsEnvSet("IOC","ioctest")
epicsEnvSet("TOP","C:/Users/'user'/testioc")
epicsEnvSet("EPICS_BASE","C:/Users/'user'/base-R7.0.4.1")
cd "C:/Users/'user'/testioc"
## Register all support components
dbLoadDatabase "dbd/test.dbd"
test_registerRecordDeviceDriver pdbbase
## Load record instances
```

(continues on next page)

(continued from previous page)

```
dbLoadRecords("db/test.db","user='user'")
cd "C:/Users/'user'/testioc/iocBoot/ioctest"
iocInit
################################################################################
## EPICS R7.0.4.1
## Rev. 2020-10-21T11:57+0200
################################################################################
iocRun: All initialization complete
## Start any sequence programs
#seq sncxxx,"user='user'"
epics>
```

Check if the database `test.db` you created is loaded correctly:

```
epics> dbl
test:pv1
test:pv2
test:add
```

As you can see 3 process variable is loaded and available. Keep this terminal open and running. Test this process variable using another terminals.

Open another shell for monitoring `test:add`:

```
>camonitor test:add
test:add                       2020-10-23 13:39:14.795006 100
```

That terminal will monitor the PV `test:add` continuously. If any value change is detected, it will be updated in this terminal. Keep it open to observe the behaviour.

Open a third shell. Using caput, modify the values of `test:pv1` and `test:pv2` as we have done in the temperature example above. You will see changes of their sum in the second terminal accordingly.

At this point, you have one IOC `testioc` running, which loaded the database `test.db` with 3 records. From other processes, you can connect to these records using Channel Access. If you add more process variable in `test.db`, you will have to `make` the *testioc* application again and restart the IOC to load the new version of the database.

You can also create and run IOCs like this in parallel with their own databases and process variables. Just keep in mind that each record instance has to have a unique name for Channel Access to work properly.

Setting the system environment

In order to run all EPICS commands anywhere by using their simple name and to build more EPICS modules using the same setup, you can set three environment variables for the current shell or user on the Windows system:

- EPICS_BASE
- EPICS_HOST_ARCH
- Path

Note that *running* IOCs only needs the Path to be set correctly (when using dynamic DLL builds). *Building* IOC applications needs EPICS_HOST_ARCH and benefits from EPICS_BASE being set.

Required settings for Path

The way you are building your binaries determines which paths have to be added to the Path variable.

- Static builds

  1. Add the EPICS Base binary directory for your target to be able to call the EPICS command line tools without specifying their fully qualified path.

  This setting is for convenience only and not mandatory. Your IOCs run without it.

- Dynamic (DLL) builds

  1. Add the EPICS Base binary directory for your target so that the EPICS DLLs are found and you can use the CLI tools without specifying the path.
  2. If you built your binaries using MinGW and want to use them under the Command Prompt or through a shortcut icon, add the MinGW binary directory ("C:\msys64\mingw64\bin") so that the MinGW runtime system DLLs are found. Inside the MSYS2 Bash shell, this location is included by default.

  Both settings are mandatory; the former for all builds, the latter under the stated condition.

Set environment using a batch or script from EPICS Base

EPICS Base provides script and batch files to help setting the environment for running EPICS commands and doing EPICS builds.

The `windows.bat` batch file in the folder called `startup` sets the environment if you use the Windows command prompt and compiled your EPICS Base using Visual Studio compilers with the help of Strawberry Perl. You probably will have to edit `windows.bat` to adapt it to your needs and `call` it from any Windows command prompt before doing EPICS commands or builds.

If you use the MSYS2 bash shell, you similarly need to adapt and run the `unix.sh` shell script from any bash shell prompt before doing EPICS commands or builds.

Set environment using the Windows settings

This method requires less effort and does not need something special to be executed or called from the command prompt.

Go to Start Menu, Type "environment" and select "Edit environment variables for your account". If you have Administrator rights and want to do it globally, you can also select `Edit the system environment variables`.

1. Select `Advance` tab, navigate to `Environment Variables` button. That should open editable tables of settings for Windows Environment.
2. Select `User Variable for 'user'` option, press NEW
3. Add EPICS BASE path here. In `Variable Name`, put "EPICS_BASE". For `Variable Value`, enter the location of your EPICS Base installation, e.g., "C:\msys64\home\'user'\base-R7.0.4.1"
4. Set the host architecture. In `Variable Name`, put "EPICS_HOST_ARCH". For `Variable Value`, put "windows-x64-mingw" or "windows-x64" (depending on your selection of compilers).
5. Navigate to the variable called `Path`. Press Edit.
6. If you are using the MinGW compilers and dynamic (DLL) linking, add the path for the MinGW64 DLLs. Press NEW and enter "C:\msys64\mingw64\bin". Press ok.

7. Add the path for the EPICS commands and DLLs. Press NEW and enter `%EPICS_BASE%\bin\%EPICS_HOST_ARCH%`. Press ok twice and you are done.
8. Restart the Machine and check if EPICS commands like `caget` and `camonitor` are being recognised as valid commands in any location and work.

Note that by default the MSYS2 shell does not inherit the parent environment. To change that behavior, you need to start the shell with the argument `-use-full-path`.

1.5 EPICS Dependencies on CentOS 8

Tags: beginner

Contents

1.5.1 Overview

This document describes the packages that must be installed in order to build EPICS base, synApps, and areaDetector on a new CentOS 8 system. For other Linux distributions the package manager and package names may be different, but the requirements are likely to be similar.

Add the Extra Packages for Enterprise Linux (EPEL) repository for the dnf package manager. This site has additional packages that are needed:

```
sudo dnf install epel-release
```

Enable the powertools repository by running:

```
sudo dnf config-manager --set-enabled powertools
```

Or on CentOS 9 Stream by running:

```
sudo dnf config-manager --set-enabled crb
```

1.5.2 Packages required to build EPICS base

```
sudo dnf install gcc gcc-c++ gcc-toolset-9-make readline-devel perl-ExtUtils-Install make
```

1.5.3 Packages required by the sequencer

```
sudo dnf install re2c
```

1.5.4 Packages required by epics-modules/asyn

```
sudo dnf install rpcgen libtirpc-devel
```

1.5.5 Packages required by the Canberra and Amptek support in epics-modules/mca

```
sudo dnf install libnet-devel libpcap-devel libusb-devel
```

1.5.6 Packages required by the Linux drivers in epics-modules/measComp

```
sudo dnf install libnet-devel libpcap-devel libusb-devel
```

1.5.7 Packages required by areaDetector/ADSupport/GraphicsMagick

```
sudo dnf install xorg-x11-proto-devel libX11-devel libXext-devel
```

1.5.8 Packages required by areaDetector/ADEiger

```
sudo dnf install zeromq-devel
```

1.5.9 Packages required to build aravis 7.0.2 for areaDetector/ADAravis

```
sudo dnf install ninja-build meson glib2-devel libxml2-devel gtk3-devel gstreamer1␣
↪gstreamer1-devel gstreamer1-plugins-base-devel libnotify-devel gtk-doc gobject-
↪introspection-devel
```

1.5.10 Packages required to build areaDetector/ADVimba

```
sudo dnf install glibmm24-devel
```

1.5.11 Packages required to build EDM

```
sudo dnf install giflib giflib-devel zlib-devel libpng-devel motif-devel libXtst-devel
```

1.5.12 Packages required to build MEDM

```
sudo dnf install libXt-devel motif-devel
```

1.6 Cross compiling to an old x86 Linux system

Tags: developer advanced

1.6.1 Introduction

I was given the task of developing a IOC which should run in a x86 PC with an old Linux distribution. My development machine was a x86_64 PC running Ubuntu 12.04.

EPICS does a great job cross compiling from a 64-bits host to a 32-bits target, if both have compatible versions of glibc, binutils, kernel, etc. In my case, however, my target had much older versions.

I considered two different solutions:

1. Create a Virtual Machine and install the target's Linux distribution. From the Virtual Machine, compile EPICS and my IOC, and then run the IOC in the target.
2. Build a toolchain configured for my target and use that toolchain to compile both EPICS and the IOC.

The first approach is the easiest, but compiling from inside a VM can be slow and the distribution was not very user friendly. So I took the second path, which I'll describe in this document.

1.6.2 Overview

I'm assuming you, like me, are running Ubuntu 64 bits. I'm also assuming you already have EPICS base downloaded and compiled. We will go through the steps of downloading, configuring and compiling Crosstool-NG, which will be used to generate our toolchain. Then we will download and compile a couple of libraries needed by EPICS (namely readline and ncurses). Finally, we will compile EPICS base and an example IOC to our target using the newly built toolchain.

1.6.3 Crosstool-NG

Downloading and extracting

First we get the tarball containing the source code and extract it.

```
wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.9.3.tar.bz2
tar -xvf crosstool-ng-1.9.3.tar.bz2
```

Crosstool-NG has a lot of dependencies, you might want to get them before compiling:

```
sudo apt-get install gawk bison flex texinfo automake libtool cvs libcurses5-dev build-
↪essential
```

Compiling

In order to compile:

```
cd crosstool-ng-1.9.3
./configure --local
make
```

Configuring

Before configuring Crosstool-NG, I gathered information about my target system:

Kernel Version:

```
$ uname -r
2.6.27.27
```

GCC Version:

```
$ gcc --version
gcc (GCC) 4.2.4
```

glibc version:

```
$ /lib/libc.so.6
GNU C Library stable release version 2.7, by Roland McGrath et al.
```

binutils version:

```
$ ld --version
GNU ld (Linux/GNU Binutils) 2.18.50.0.9.20080822
```

Based on this information, and on a lot of trial and error, I configured Crosstool-NG as follows (everything else set as default):

```
$ ./ct-ng menuconfig

PATHS AND MISC OPTIONS
   [*] Use obsolete features
```

(continues on next page)

(continued from previous page)

```
TARGET OPTIONS
   Target architecture (x86)
   Bitness: (32-bit)
   (i686) Architecture Level

OPERATING SYSTEM
   Target OS (linux)
   Linux kernel version (2.6.27.55)

BINARY UTILITIES
   Binutils version (2.17)

C-COMPILER
   GCC version (4.2.4)
   [*] C++

C-LIBRARY
   glibc version (2.6.1)
```

I tried other configurations, but they crashed the compilation process.

Compiling the toolchain

Now we can compile the toolchain:

```
./ct-ng build
```

This will take a while. Go get some coffee or watch a cat video on Youtube.

Once built, the toolchain will be in `$HOME/x-tools/i686-unknown-linux-gnu/`

It's a good idea (I think) to put the cross-compiler binaries on your path. Add this to the end of your ~/.bashrc:

```
PATH=$PATH:$HOME/x-tools/i686-unknown-linux-gnu/bin
```

Then source your .bashrc, so the changes take effect.

```
. ~/.bashrc
```

1.6.4 EPICS dependencies

In order to properly build epics-base to our target system, we have to take care of EPICS dependencies first. Namely, the libraries 'readline' and 'ncurses'.

They will be installed in our toolchain directory. We have to make it writable:

```
chmod -R +w $HOME/x-tools/i686-unknown-linux-gnu/i686-unknown-linux-gnu/sys-root/usr
```

Now we can download, configure, compile and install the libraries.

readline

```
wget ftp://ftp.cwru.edu/pub/bash/readline-6.2.tar.gz
tar -xvf readline-6.2.tar.gz
cd readline-6.2
./configure --prefix=$HOME/x-tools/i686-unknown-linux-gnu/i686-unknown-linux-gnu/sys-
↪root/usr --host=i686-unknown-linux-gnu
make
make install
```

ncurses

```
wget ftp://ftp.gnu.org/pub/gnu/ncurses/ncurses-5.9.tar.gz
tar -xvf ncurses-5.9.tar.gz
cd ncurses-5.9
./configure --prefix=$HOME/x-tools/i686-unknown-linux-gnu/i686-unknown-linux-gnu/sys-
↪root/usr --host=i686-unknown-linux-gnu
make
make install
```

1.6.5 Configure cross-compilation in EPICS

We're almost done. Back to the epics-base directory, open the file: `$EPICS_BASE/configure/CONFIG_SITE`

Change the line:

```
CROSS_COMPILER_TARGET_ARCHS=
```

To:

```
CROSS_COMPILER_TARGET_ARCHS=linux-686
```

This tells EPICS base to be compiled both for the host system and for the linux-686 target.

Save and close. Now we will create our own target configuration file, based on a existing file.

```
cd $EPICS_BASE/configure/os
cp CONFIG_SITE.Common.linux-x86 CONFIG_SITE.Common.linux-686
```

Open `CONFIG_SITE.Common.linux-686` and edit it. Comment out the line:

```
#COMMANDLINE_LIBRARY = READLINE
```

And uncomment:

```
COMMANDLINE_LIBRARY = READLINE_NCURSES
```

At the end of the file, add the lines:

```
GNU_DIR=$HOME/x-tools/i686-unknown-linux-gnu
GNU_TARGET=i686-unknown-linux-gnu
```

This tells EPICS to search for both readline and ncurses libraries during compilation. The last two lines indicate the location of the toolchain and its prefix. Save and close. Last file to edit: CONFIG.Common.linux-686

Change the line that says

```
VALID_BUILDS = Ioc
```

To

```
VALID_BUILDS = Host Ioc
```

This is needed in order to get caRepeater compiled, according to this source.

Recompile EPICS base

Now, we recompile EPICS base:

```
make clean uninstall
make
```

Hopefully, we have everything in place to start developing our IOC's.

1.6.6 Example IOC

Let's create a working directory for our programs. I decided to call it 'apps':

```
mkdir ~/apps
```

Creating

To create an example IOC, we use a Perl script provided by EPICS:

```
cd ~/apps
mkdir myexample
cd myexample
makeBaseApp.pl -t example myexample
```

The last command tells the script to create an application named 'myexample' using the template (option -t) 'example'. Now we make our IOC bootable

```
makeBaseApp.pl -i -t example myexample
```

It will ask you what the target is. We went to all this trouble to be able to select:

```
linux-686
```

For the Application Name, just hit enter.

Configuring

Add this line to the file ~/apps/myexample/configure/CONFIG_SITE

```
STATIC_BUILD=YES
```

This will statically link EPICS libraries into our executable.

Now, let's consider that you will put your IOC in the folder /home of your target system. Edit the file ~/apps/myexample/iocBoot/iocmyexample/envPaths, so it will be:

```
epicsEnvSet("ARCH","linux-686")
epicsEnvSet("IOC","iocskel")
epicsEnvSet("TOP","/home/myexample")
epicsEnvSet("EPICS_BASE","/home/myexample")
```

Note that we set EPICS base to coincide with the folder of our IOC. I did it because the IOC depends on the caRepeater program, which would be present in an EPICS base if we had one in our target. Because we don't, I'll simply copy the caRepeater generated by the host to the the /bin folder of our IOC folder:

```
cp $EPICS_BASE/bin/linux-686/caRepeater ~/apps/myexample/bin/linux-686/
```

Compiling

Compile the IOC and prepare it for execution.

```
make
chmod +x iocBoot/iocmyexample/st.cmd
```

Note that you won't be able to run the IOC in your host system, given that it was compiled to our target system. You won't be able to run it in your target system neither, as your target lacks three libraries: two needed by caRepeater and one needed by the IOC.

First, take care of the libraries needed by caRepeater:

```
mkdir ~/apps/myexample/lib
mkdir ~/apps/myexample/lib/linux-686/
cp $EPICS_BASE/lib/linux-686/libca.so.3.14 ~/apps/myexample/lib/linux-686
cp $EPICS_BASE/lib/linux-686/libCom.so.3.14 ~/apps/myexample/lib/linux-686
```

Then copy libreadline from your host's folder:

```
~/x-tools/i686-unknown-linux-gnu/i686-unknown-linux-gnu/sys-root/usr/lib/libreadline.so.
↪6.2
```

To your target's folder:

```
/lib/libreadline.so.6
```

Please note the change in the filename.

Executing

After copying your myexample folder to your target's /home folder, you can run your IOC:

```
cd /home/myexample/iocBoot/iocmyexample
./st.cmd
```

If everything goes as expected, you will have an epics prompt:

```
epics>
```

Try listing the records:

```
epics> dbl
bruno:ai1
bruno:ai2
bruno:ai3
bruno:aiExample
bruno:aiExample1
bruno:aiExample2
bruno:aiExample3
bruno:aSubExample
bruno:calc1
bruno:calc2
bruno:calc3
bruno:calcExample
bruno:calcExample1
bruno:calcExample2
bruno:calcExample3
bruno:compressExample
bruno:subExample
bruno:xxxExample
```

1.7 Creating an IOC Application

Tags: user developer

This example shows how to create an IOC Application with an IOC using StreamDevice to talk to devices, e.g., via ethernet.

Create a directory for the IOC Applications. For example `$HOME/EPICS/IOCs`

```
cd $HOME/EPICS
mkdir IOCs
cd IOCs
```

Create a top for an IOC called `sampleIOC`

```
mkdir sampleIOC; cd sampleIOC
makeBaseApp.pl -t example sampleIOC
makeBaseApp.pl -i -t example sampleIOC
Using target architecture darwin-x86 (only one available)
The following applications are available:
```

(continues on next page)

(continued from previous page)

```
sampleIOC
What application should the IOC(s) boot?
The default uses the IOC's name, even if not listed above.
Application name? (just return)
```

Now, by running `make`, a sample IOC like the demo/test IOC is built. Next, we want to add asyn and StreamDevice to the IOC. For this, we add the stream and asyn libraries to the Makefile. Edit `sampleIOCApp/src/Makefile` and add the block

```
#add asyn and streamDevice to this IOC production libs
sampleIOC_LIBS += stream
sampleIOC_LIBS += asyn
```

The application must also load `asyn.dbd` and `stream.dbd` to use StreamDevice. This can be put into a generated dbd, e.g into `xxxSupport.dbd` which already gets included by the Makefile. So the `xxxSupport.dbd` now reads:

```
cat sampleIOCApp/src/xxxSupport.dbd
include "xxxRecord.dbd"
device(xxx,CONSTANT,devXxxSoft,"SoftChannel")
#
include "stream.dbd"
include "asyn.dbd"
registrar(drvAsynIPPortRegisterCommands)
registrar(drvAsynSerialPortRegisterCommands)
registrar(vxi11RegisterCommands)
```

To find the dbd files, you have to add the paths to these files in `configure/RELEASE`:

```
...
# Build variables that are NOT used in paths should be set in
# the CONFIG_SITE file.
# Variables and paths to dependent modules:
SUPPORT = ${HOME}/EPICS/support
ASYN=$(SUPPORT)/asyn
STREAM=$(SUPPORT)/stream
# If using the sequencer, point SNCSEQ at its top directory:
#SNCSEQ = $(MODULES)/seq-ver
...
```

If `make` was done before, `make distclean` is probably required. Anyway, then `make`. The newly created IOC can be run with:

```
cd iocBoot/iocsampleIOC/
chmod u+x st.cmd
./st.cmd
```

Not very interesting yet, because there is no database file nor a protocol file.

```
ls -la sampleIOCApp/Db/
total 56
drwxr-xr-x 11 maradona  staff   374  Jun  1  16:47  .
drwxr-xr-x  5 maradona  staff   170  Jun  1  12:46  ..
-rw-r--r--  1 maradona  staff   523  Jun  1  12:46  Makefile
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x  2 maradona  staff    68  Jun  1  16:47  O.Common
drwxr-xr-x  3 maradona  staff   102  Jun  1  16:47  O.darwin-x86
-rw-r--r--  1 maradona  staff  1761  Jun  1  12:46  circle.db
-rw-r--r--  1 maradona  staff  1274  Jun  1  12:46  dbExample1.db
-rw-r--r--  1 maradona  staff   921  Jun  1  12:46  dbExample2.db
-rw-r--r--  1 maradona  staff   286  Jun  1  12:46  dbSubExample.db
-rw-r--r--  1 maradona  staff   170  Jun  1  12:46  sampleIOCVersion.db
-rw-r--r--  1 maradona  staff   307  Jun  1  12:46  user.substitutions
```

Note that this is a `Db` directory and not the `db` directory that is in `./sampleIOC`. For MDOxxxx scopes by Tektronix, the database (`.db`) and protocol (`.proto`) file can look something like

```
cat MDO.db
record(stringin, $(P)$(R)idn){
    field(DESC, "Asks for info blabla")
    field(DTYP, "stream")
    field(INP, "@MDO.proto getStr(*IDN,99) $(PORT) $(A)")
    field(PINI, "YES")
}

cat MDO.proto
Terminator = LF;
getStr{
    out "$1?";
    in "%s";
    @replytimeout {out "$1?"; in "%s";}
}
```

Now, we add to `sampleIOCApp/Db/Makefile` the information that these files must be included in the compilation. So

```
cat sampleIOCApp/Db/Makefile
TOP=../..
include $(TOP)/configure/CONFIG
#----------------------------------------
# ADD MACRO DEFINITIONS BELOW HERE
# Install databases, templates & substitutions like this
DB += circle.db
DB += dbExample1.db
DB += dbExample2.db
DB += sampleIOCVersion.db
DB += dbSubExample.db
DB += user.substitutions
DB += MDO.db
DB += MDO.proto
# If .db template is not named *.template add
# _TEMPLATE =
include $(TOP)/configure/RULES
#----------------------------------------
# ADD EXTRA GNUMAKE RULES BELOW HERE
```

Again, `make` in directory sampleIOC. Finally, we add IP port configuration, setting the Stream path and loading the database to the `st.cmd` file. The `st.cmd` should read:

```
cat st.cmd

#!../../bin/darwin-x86/sampleIOC

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

< envPaths

epicsEnvSet ("STREAM_PROTOCOL_PATH","$(TOP)/db")

cd "${TOP}"

## Register all support components
dbLoadDatabase "dbd/sampleIOC.dbd"
sampleIOC_registerRecordDeviceDriver pdbbase

## Load record instances
dbLoadTemplate "db/user.substitutions"
dbLoadRecords "db/sampleIOCVersion.db", "user=UUUUUU"
dbLoadRecords "db/dbSubExample.db", "user=UUUUUU"

#IF if the user also defines EPICS_CAS_INTF_ADDR_LIST then beacon address
#list automatic configuration is constrained to the network interfaces specified
#therein, and therefore only the broadcast addresses of the specified LAN interfaces,
#and the destination addresses of all specified point-to-point links, will be␣
↪automatically configured.
#epicsEnvSet ("EPICS_CAS_INTF_ADDR_LIST","aaa.aaa.aaa.aaa")

# connect to the device ... IP-Address ! Port 2025 used by textronix, see manual
drvAsynIPPortConfigure("L0","bbb.bbb.bbb.bbb:pppp",0,0,0)

## Load record instances
dbLoadRecords("db/MDO.db", "P=UUUUUU:,PORT=L0,R=MDO:,L=0,A=0")

#- Set this to see messages from mySub
#var mySubDebug 1

#- Run this to trace the stages of iocInit
#traceIocInit

cd "${TOP}/iocBoot/${IOC}"
iocInit

## Start any sequence programs
#seq sncExample, "user=UUUUUU"
```

In here, you have to replace *UUUUUU* with the user name that runs the EPICS IOC (you?). *bbb.bbb.bbb.bbb* is the IP of the device (e.g. the scope) and *pppp* the port on which it listens. *EPICS_CAS_INTF_ADDR_LIST* can be used if there are two network interfaces (e.g. wlan and eth0).

The following commands might be necessary with multiple network interfaces:

```
export EPICS_CA_ADDR_LIST=ccc.ccc.ccc.ccc << Broadcast address of the network
export EPICS_CA_AUTO_ADDR_LIST=NO
```

1.8 EPICS applications on Mac OS X

1.8.1 How do I get EPICS applications to work with a Mac OS X firewall?

These instructions apply to OS X 10.6 and higher

- Start the System Preferences application
- Select the "Security & Privacy" pane
- Select the "Firewall" tab
- Open the lock to allow changes
- Ensure that the firewall is on
- Click the "Advanced" button
- Add your EPICS applications (e.g. caRepeater, caget, camonitor, caput, StripTool, edm, soft IOCs, etc.) to the list and set the firewall to "Allow incoming connections" to them

1.9 Configuring vxWorks 6.x

Tags: developer advanced

1.9.1 vxWorks 6.x Information

This page provides a advice on configuring and using the Wind River's Workbench environment and the vxWorks 6.x RTOS with EPICS. If you discover any other information that ought to be published here, please let me know.

Note that there is a *separate page* provided for users of vxWorks 5.x and Wind River's Tornado.

Tornado 2.2 and Linux

PowerPC

Configuring WRS Tornado 2.x for EPICS

Configuring a vxWorks 6.x image

Using the Wind River Workbench to create a vxWorks image suitable for running EPICS IOCs, the following components are **required** in addition to the standard components included with a new vxWorks 6.x Image Project (System Image) with a PROFILE_DEVELOPMENT Configuration Profile:

- C++ components
  - standard library
    - C++ Iostreams and other ... — INCLUDE_CPLUS_IOSTREAMS
- Network Components (default)
  - Network Applications (default)

        * SNTP Components
            - SNTP Client (daemon) — INCLUDE_IPSNTPC
              Set the NTP server addresses under here. The primary server IPv4 address can be set to sysBootParams.had for the IOC to always use its boot host as an NTP server.
    - Network Core Components (default)
        * Backwards compatibility wrapper routines
            - libc wrappers
            - sntpcTimeGet wrapper — INCLUDE_IPWRAP_SNTPCTIMEGET
        * network init — INCLUDE_NET_INIT
- operating system components (default)
    - IO system components (default)
        * IO Subsystem Components
            - Basic IO System
            - max # open files in the system — NUM_FILES
              Configure this to more than the maximum number of CA sessions you expect need to connect into and out of this IOC at the same time. The CA protocol uses one file handle per client, and every additional network socket, serial port and other vxWorks device will use at least one.
    - kernel components (default)
        * unix compatable environment variables (default)
            - install environment variable task create/delete hooks — ENV_VAR_USE_HOOKS
              This variable must be set to FALSE.

The following components are **optional** but will often be wanted:

- Network Components (default)
    - Network Applications (default)
        * SNTP Components
            - INCLUDE_IPSNTP_CMD
        * DNS Client — INCLUDE_IPDNSC
          Set the DNS domain name and at least the DNS primary name server under here. The server can be set to sysBootParams.had for the IOC to always use its boot host as a DNS server
    - Network Core Components (default)
        * Backwards compatibility wrapper routines
            - libc wrappers
            - arp utility wrapper — INCLUDE_IPWRAP_ARP
            - utilslib wrappers
            - ifShow wrapper — INCLUDE_IPWRAP_IFSHOW
            - ifconfig wrapper — INCLUDE_IPWRAP_IFCONFIG
            - netstat wrapper — INCLUDE_IPWRAP_NETSTAT
            - ping wrapper — INCLUDE_IPWRAP_PING
            - routec wrapper — INCLUDE_IPWRAP_ROUTECMD

- development tool components (default)
  - spy — INCLUDE_SPY
- operating system components (default)
  - IO system components (default)
    - NFS Components
      - NFS client All — INCLUDE_NFS_CLIENT_ALL

These components are included in the PROFILE_DEVELOPMENT configuration by default but **not required** by EPICS so may safely be excluded:

- Network Components (default)
  - Network Core Components (default)
    - Backwards compatibility wrapper routines
      - libc wrappers
      - getservbyname wrapper — INCLUDE_IPWRAP_GETSERVBYNAME
      - getservbyport wrapper — INCLUDE_IPWRAP_GETSERVBYPORT
- application components (default)
  - application initialization — INCLUDE_USER_APPL
- development tool components (default)
  - Compiler support routines
    - Diab compiler support routines — INCLUDE_DIAB_INTRINSICS
- operating system components (default)
  - ANSI C components (libc) (default)
    - ANSI locale — INCLUDE_ANSI_LOCALE
    - ANSI stdio extensions — INCLUDE_ANSI_STDIO_EXTRA
  - POSIX components
    - POSIX timers (default) — INCLUDE_POSIX_TIMERS
    - sigevent notification library — INCLUDE_SIGEVENT
  - Real Time Process components — FOLDER_RTP
  - SYSCTL Component — FOLDER_SYSCTL

vxWorks 6.6 GNU Header stdexcept

There is a bug in the GNU C++ header file stdexcept as delivered with vxWorks 6.6 which results in some undefined symbols when you try to load the IOC code. The header has been fixed in later vxWorks releases, and there may have been an official Wind River patch issued to fix this, but Erik Bjorklund has provided this patch to address the problem.

Adding a CR/CSR Master Window to the mv6100 BSP

Eric Bjorklund gave a talk at a EPICS collaboration meeting in June 2006 describing how he added support for accessing the VME CR/CSR address space to the mv6100 BSP.

1.10 Configuring Tornado/vxWorks 5.5.x

Tags: user developer advanced

1.10.1 Tornado/vxWorks 5.5.x Information

This page provides a repository for information about using the WRS Tornado environment and the vxWorks 5.5.x RTOS with EPICS that doesn't really belong anywhere else on this site.

Note that there is a *separate page* provided for users of vxWorks 6.x and Wind River Workbench.

There is also a reasonably good Tornado 2.0 FAQ available on the web, mostly comprising answers to questions posted to the comp.os.vxworks news group.

1.10.2 Tornado 2.2 (vxWorks 5.5.x)

Installation

According to the Tornado 2.2 release notes, you cannot install multiple host and/or target architectures in the same directory.

Linux Hosting

See this page for information building vxWorks target code on Linux.

EPICS Support

VxWorks 5.5 is only supported on EPICS 3.14.x and 3.15.x releases. From Base-3.16 and later you must use VxWorks 6.6 or later.

1.10.3 PowerPC Issues

There is a separate page discussing the specific problems associated with using PowerPC CPUs under vxWorks/Tornado.

1.11 Common Database patterns

Tags: developer

1.11.1 Pull Alarm Status w/o Data

This is useful to bring alarm status in without affecting the value stored in a record. In the following example the alarm status of **$(P):set** is fetched by **$(P):rbv** when it is processed, but the value is read from a different record.

```
record(bo, "$(P):set") {
    field(OSEV, "MAJOR")
    field(FLNK, "$(P):rbv")
}
```

```
record(bi, "$(P):rbv") {
    field(SDIS, "$(P):set NPP MS")
    field(DISV, "-1")
    field(INP , "$(P):some:other:record")
}
```

1.11.2 Combined Setting and Readback

Use when a single PV is desired. Could be used to show the results of rounding in a float to fixed precision conversion.

In the following example the value read from a 'bi' is inverted so that the associated 'bo' acts as a toggle.

```
record(bi, "$(P):rbv") {
    field(DTYP, "...")
    field(INP , "...")
    field(PINI, "YES")
    field(FLNK, "$(P):inv")
}
```

```
record(calcout, "$(P):inv")
    field(CALC, "!A")
    field(INPA, "$(P):rbv")
    field(OUT , "$(P) NPP")
}
```

```
record(bo, "$(P)") {
    field(DTYP, "...")
    field(OUT , "...")
    field(FLNK, "$(P):rbv")
}
```

The important point is the NPP modifier on output link of the 'calcout' record. This updates the VAL field of the 'bo' record (and posts monitors) without processing it.

1.12 How to avoid copying arrays with waveformRecord

Tags: developer

1.12.1 Introduction

This page describes how to use the array field memory management. This allows array data to be moved into and out of the value (aka BPTR) field of the `waveform`, `aai`, and `aao` types.

Making use of this feature involves replacing the pointer stored in the BPTR field with another (user allocated) pointer. The basic rules are:

1. BPTR, and the memory it is currently pointing to, can only be accessed while the record is locked.
2. NELM may not be changed.
3. BPTR must always point to a piece of memory large enough to accommodate the maximum number of elements (as given by the NELM field).

Rule #1 means that it is only safe to read, write, or de-reference the BPTR field from a device support function, or after manually calling `dbScanLock()`. Rule #3 means that BPTR can never be set to NULL, and when replacing BPTR, the replacement must be allocated large enough for the worst case. An external client may put an array of up to NELM elements to the field at almost any time.

1.12.2 Example

```
/* Demonstration of using custom allocation for waveformRecord buffers.
 *
 * Requires EPICS Base with the array field memory management patch
 * [https://code.launchpad.net/~epics-core/epics-base/array-opt](https://code.launchpad.
↪net/%7Eepics-core/epics-base/array-opt)
 *
 * This example makes inefficient use of malloc() and
 * free().  This is done to make clear where new memory appears.
 * In reality a free list should be used.
 *
 * Also be aware that this example will use 100% of the time of one CPU core.
 * However, this will be spread across available cores.
 *
 * To use this example include the following in a DBD file:
 *
 * device(waveform,CONSTANT,devWfZeroCopy,"Zero Copy Demo")
 *
 * Also include a record instance
 *
 * record(waveform, "$(NAME)") {
 *  field(DTYP, "Zero Copy Demo")
 *  field(FTVL, "SHORT")
 *  field(NELM, "100")
 *  field(SCAN, "I/O Intr")
 * }
 */
```

(continues on next page)

(continued from previous page)

```
#include <errlog.h>
#include <initHooks.h>
#include <ellLib.h>
#include <devSup.h>
#include <dbDefs.h>
#include <dbAccess.h>
#include <cantProceed.h>
#include <epicsTypes.h>
#include <epicsMutex.h>
#include <epicsEvent.h>
#include <epicsThread.h>
#include <menuFtype.h>
#include <dbScan.h>

#include <waveformRecord.h>

static ELLLIST allPvt = ELLLIST_INIT;

struct devicePvt {
    ELLNODE node;

    /* synchronize access to this structure */
    epicsMutexId lock;
    /* wakeup the worker when another update is needed */
    epicsEventId wakeup;
    /* notify the scanner thread when another update is available */
    IOSCANPVT scan;

    /* the next update */
    void *nextBuffer;
    epicsUInt32 maxbytes, numbytes;
};

static void startWorkers(initHookState);

static long init(int phase)
{
    if(phase!=0)
        return 0;
    initHookRegister(&startWorkers);
    return 0;
}

static long init_record(waveformRecord *prec)
{
    struct devicePvt *priv;
    if(prec->ftvl!=menuFtypeSHORT) {
        errlogPrintf("%s.FTVL must be set to SHORT for this example\n", prec->name);
        return 0;
    }

    /* cleanup array allocated by record support.
```

(continues on next page)

(continued from previous page)

```
     * Not necessary since we use calloc()/free(),
     * but needed when allocating in other ways.
     */
    free(prec->bptr);
    prec->bptr = callocMustSucceed(prec->nelm, dbValueSize(prec->ftvl), "first buf");

    priv = callocMustSucceed(1, sizeof(*priv), "init_record devWfZeroCopy");
    priv->lock = epicsMutexMustCreate();
    priv->wakeup = epicsEventMustCreate(epicsEventFull);
    scanIoInit(&priv->scan);
    priv->maxbytes = prec->nelm*dbValueSize(prec->ftvl);

    ellAdd(&allPvt, &priv->node);

    prec->dpvt = priv;
    return 0;
}

static void worker(void*);

static void startWorkers(initHookState state)
{
    ELLNODE *cur;
    /* Don't start worker threads until
     * it is safe to call scanIoRequest()
     */
    if(state!=initHookAfterInterruptAccept)
        return;
    for(cur=ellFirst(&allPvt); cur; cur=ellNext(cur))
    {
        struct devicePvt *priv = CONTAINER(cur, struct devicePvt, node);
        epicsThreadMustCreate("wfworker",
                              epicsThreadPriorityHigh,
                              epicsThreadGetStackSize(epicsThreadStackSmall),
                              &worker, priv);
    }
}

static void worker(void* raw)
{
    struct devicePvt *priv=raw;
    void *buf = NULL;
    epicsUInt32 nbytes = priv->maxbytes;

    while(1) {

        if(!buf) {
            /* allocate and initialize a new buffer for later (local) use */
            size_t i;
            epicsInt16 *ibuf;
            buf = callocMustSucceed(1, nbytes, "buffer");
            ibuf = (epicsInt16*)buf;
```

(continues on next page)

(continued from previous page)

```
            for(i=0; i<nbytes/2; i++)
            {
                ibuf[i] = rand();
            }
        }

        /* wait for Event signal when record is scanning 'I/O Intr',
         * and timeout when record is scanning periodic
         */
        if(epicsEventWaitWithTimeout(priv->wakeup, 1.0)==epicsEventError) {
            cantProceed("worker encountered an error waiting for wakeup\n");
        }

        epicsMutexMustLock(priv->lock);

        if(!priv->nextBuffer) {
            /* make the local buffer available to the read_wf function */
            priv->nextBuffer = buf;
            buf = NULL;
            priv->numbytes = priv->maxbytes;
            scanIoRequest(priv->scan);
        }

        epicsMutexUnlock(priv->lock);
    }
}

static long get_iointr_info(int dir, dbCommon *prec, IOSCANPVT *scan)
{
    struct devicePvt *priv=prec->dpvt;
    if(!priv)
        return 0;
    *scan = priv->scan;
    /* wakeup the worker when this thread is placed in the I/O scan list */
    if(dir==0)
        epicsEventSignal(priv->wakeup);
    return 0;
}

static long read_wf(waveformRecord *prec)
{
    struct devicePvt *priv=prec->dpvt;
    if(!priv)
        return 0;

    epicsMutexMustLock(priv->lock);

    if(priv->nextBuffer) {
        /* an update is available, so claim it. */

        if(prec->bptr)
            free(prec->bptr);
```

(continues on next page)

(continued from previous page)

```
        prec->bptr = priv->nextBuffer; /* no memcpy! */
        priv->nextBuffer = NULL;
        prec->nord = priv->numbytes / dbValueSize(prec->ftvl);

        epicsEventSignal(priv->wakeup);
    }

    epicsMutexUnlock(priv->lock);

    assert(prec->bptr);

    return 0;
}

static
struct dset5 {
    dset com;
    DEVSUPFUN read;
} devWfZeroCopy = {
{5, NULL,
 &init,
 &init_record,
 &get_iointr_info
},
 &read_wf
};

#include <epicsExport.h>

epicsExportAddress(dset, devWfZeroCopy);
```

1.13 Application Developer's Guide

Tags: developer advanced

The classic Application Developer's Guide as an online document. At this stage, this contains only the descriptive sections of the document, detailed API documentation will be generated from the source code with Doxygen.

1.13.1 Getting Started

Tags: beginner user developer

Introduction

This chapter provides a brief introduction to creating EPICS IOC applications. It contains:

- Instructions for creating, building, and running an example IOC application.
- Instructions for creating, building, and executing example Channel Access clients.
- Briefly describes iocsh, which is a base supplied command shell.
- Describes rules for building IOC components.
- Describes makeBaseApp.pl, which is a perl script that generates files for building applications.
- Briefly discusses vxWorks boot parameters

This chapter will be hard to understand unless you have some familarity with IOC concepts such as record types, device and driver support and have had some experience with creating ioc databases. Once you have this experience, this chapter provides most of the information needed to build applications. The example that follows assumes that EPICS base has already been built.

Example IOC Application

This section explains how to create an example IOC application in a directory `<top>`, naming the application `myexampleApp` and the ioc directory `iocmyexample`.

Check that `EPICS_HOST_ARCH` is defined

Execute the command:

```
echo $EPICS_HOST_ARCH
```

or

```
set EPICS_HOST_ARCH
```

This should display your workstation architecture, for example `linux-x86` or `win32-x86`. If you get an "Undefined variable" error, you should set `EPICS_HOST_ARCH` to your host operating system followed by a dash and then your host architecture, e.g. `solaris-sparc`. The perl script `EpicsHostArch.pl` in the base/startup directory has been provided to help set `EPICS_HOST_ARCH`.

Create the example application

The following commands create an example application.

```
mkdir <top>
cd <top>
<base>/bin/<arch>/makeBaseApp.pl -t example myexample
<base>/bin/<arch>/makeBaseApp.pl -i -t example myexample
```

Here, <arch> indicates the operating system architecture of your computer. For example, solaris-sparc. The last command will ask you to enter an architecture for the IOC. It provides a list of architectures for which base has been built.

The full path name to <base> (an already built copy of EPICS base) must be given. Check with your EPICS system administrator to see what the path to your <base> is. For example:

```
/home/phoebus/MRK/epics/base/bin/linux-x86/makeBaseApp.pl ...
```

Windows Users Note: Perl scripts must be invoked with the command perl <scriptname> on Windows. Perl script names are case sensitive. For example to create an application on Windows:

```
perl C:\epics\base\bin\win32-x86\makeBaseApp.pl -t example myexample
```

Inspect files

Spend some time looking at the files that appear under <top>. Do this *before* building. This allows you to see typical files which are needed to build an application without seeing the files generated by make.

Sequencer Example

The sequencer is now supported as an unbundled product. The example includes an example state notation program, sncExample.stt. As created by makeBaseApp the example is not built or executed.

Before sncExample.stt can be compiled, the sequencer module must have been built using the same version of base that the example uses.

To build sncExample edit the following files:

- configure/RELEASE – Set SNCSEQ to the location of the sequencer.
- iocBoot/iocmyexample/st.cmd – Remove the comment character # from this line:

  #seq sncExample, "user=<user>"

The Makefile contains commands for building the sncExample code both as a component of the example IOC application and as a standalone program called sncProgram, an executable that connects through Channel Access to a separate IOC database.

Build

In directory <top> execute the command

```
make
```

NOTE: On systems where GNU make is not the default another command is required, e.g. gnumake, gmake, etc. See you EPICS system administrator.

Inspect files

This time you will see the files generated by make as well as the original files.

Run the ioc example

The example can be run on vxWorks, RTEMS, or on a supported host.

- On a host, e.g. Linux or Solaris

```
cd <top>/iocBoot/iocmyexample
../../bin/linux-x86/myexample st.cmd
```

- vxWorks/RTERMS – Set your boot parameters as described at the end of this chapter and then boot the ioc.

After the ioc is started try some of the shell commands (e.g. `dbl` or `dbpr <recordname>`) described in the chapter "IOC Test Facilities". In particular run `dbl` to get a list of the records.

The iocsh command interpreter used on non-vxWorks IOCs provides a help facility. Just type:

```
help
```

or

```
help <cmd>
```

where <cmd> is one of the commands displayed by help. The help command accepts wildcards, so

```
help db*
```

will provide information on all commands beginning with the characters db. On vxWorks the help facility is available by first typing:

```
iocsh
```

Channel Access Host Example

An example host example can be generated by:

```
cd <mytop>
<base>/bin/<arch>/makeBaseApp.pl -t caClient caClient
make
```

(or gnumake, as required by your operating system)

Two channel access examples are provided:

caExample
: This example program expects a pvname argument, connects and reads the current value for the pv, displays the result and terminates. To run this example just type.

<mytop>/bin/<hostarch>/caExample <pvname> where

- <mytop> is the full path name to your application top directory.
- <hostarch> is your host architecture.

- <pvname> is one of the record names displayed by the dbl ioc shell command.

caMonitor
: This example program expects a filename argument which contains a list of pvnames, each appearing on a separate line. It connects to each pv and issues monitor requests. It displays messages for all channel access events, connection events, etc.

iocsh

Because the vxWorks shell is only available on vxWorks, EPICS base provides iocsh. In the main program it can be invoked as follows:

```
iocsh("filename")
```

or

```
iocsh(0)
```

If the argument is a filename, the commands in the file are executed and iocsh returns. If the argument is 0 then iocsh goes into interactive mode, i.e. it prompts for and executes commands until an exit command is issued.

This shell is described in more detail in Chapter *[chap:IOC Shell]*, "IOC Shell".

On vxWorks iocsh is not automatically started. It can be started by just giving the following command to the vxWorks shell.

```
iocsh
```

To get back to the vxWorks shell just say

```
exit
```

Building IOC components

Detailed build rules are given in chapter *Build Facility*. This section describes methods for building most components needed for IOC applications. It uses excerpts from the myexampleApp/src/Makefile that is generated by makeBaseApp.

The following two types of applications can be built:

1. Support applications

These are applications meant for use by ioc applications. The rules described here install things into one of the following directories that are created just below <top>:

include
: C include files are installed here. Either header files supplied by the application or header files generated from xxxRecord.dbd or xxxMenu.dbd files.

dbd
: Each file contains some combination of include, recordtype, device, driver, and registrar database definition commands. The following are installed:

- xxxRecord.dbd and xxxMenu.dbd files
- An arbitrary xxx.dbd file
- ioc applications install a file yyy.dbd generated from file yyyInclude.dbd.

db
: Files containing record instance definitions.

lib/<arch>
: All source modules are compiled and placed in shared or static library (win32 dll)

2. IOC applications

These are applications loaded into actual IOCs.

Binding to IOC components

Because many IOC components are bound only during ioc initialization, some method of linking to the appropriate shared and/or static libraries must be provided. The method used for IOCs is to generate, from an `xxxInclude.dbd` file, a C++ program that contains references to the appropriate library modules. The following database definitions keywords are used for this purpose:

```
recordtype
device
driver
function
variable
registrar
```

The method also requires that IOC components contain an appropriate epicsExport statement. All components must contain the statement:

```
#include <epicsExport.h>
```

Any component that defines any exported functions must also contain:

```
#include <registryFunction.h>
```

Each record support module must contain a statement like:

```
epicsExportAddress(rset,xxxRSET);
```

Each device support module must contain a statement like:

```
epicsExportAddress(dset,devXxxSoft);
```

Each driver support module must contain a statement like:

```
epicsExportAddress(drvet,drvXxx);
```

Functions are registered using an `epicsRegisterFunction` macro in the C source file containing the function, along with a `function` statement in the application database description file. The makeBaseApp example thus contains the following statements to register a pair of functions for use with a subroutine record:

```
epicsRegisterFunction(mySubInit);
epicsRegisterFunction(mySubProcess);
```

The database definition keyword `variable` forces a reference to an integer or double variable, e.g. debugging variables. The `xxxInclude.dbd` file can contain definitions like:

```
variable(asCaDebug,int)
variable(myDefaultTimeout,double)
```

The code that defines the variables must include code like:

```
int asCaDebug = 0;
epicsExportAddress(int,asCaDebug);
```

The keyword `registrar` signifies that the epics component supplies a named registrar function that has the prototype:

```
typedef void (*REGISTRAR)(void);
```

This function normally registers things, as described in Chapter *[Registry]*, "Registry" on page . The makeBaseApp example provides a sample iocsh command which is registered with the following registrar function:

```
static void helloRegister(void) {
    iocshRegister(&helloFuncDef, helloCallFunc);
}
epicsExportRegistrar(helloRegister);
```

Makefile rules

Building a support application.

```
# xxxRecord.h will be created from xxxRecord.dbd
DBDINC += xxxRecord
DBD += myexampleSupport.dbd

LIBRARY_IOC += myexampleSupport

myexampleSupport_SRCS += xxxRecord.c
myexampleSupport_SRCS += devXxxSoft.c
myexampleSupport_SRCS += dbSubExample.c

myexampleSupport_LIBS += $(EPICS_BASE_IOC_LIBS)
```

The `DBDINC` rule looks for a file `xxxRecord.dbd`. From this file a file `xxxRecord.h` is created and installed into `<top>/include`

The `DBD` rule finds `myexampleSupport.dbd` in the source directory and installs it into `<top>/dbd`

The `LIBRARY_IOC` variable requests that a library be created and installed into `<top>/lib/<arch>`

The `myexampleSupport_SRCS` statements name all the source files that are compiled and put into the library.

The above statements are all that is needed for building many support applications.

Building the IOC application

The following statements build the IOC application:

```
PROD_IOC = myexample

DBD += myexample.dbd

# myexample.dbd will be made up from these files:
myexample_DBD += base.dbd
myexample_DBD += xxxSupport.dbd
myexample_DBD += dbSubExample.dbd

# <name>_registerRecordDeviceDriver.cpp will be created from <name>.dbd
myexample_SRCS += myexample_registerRecordDeviceDriver.cpp
myexample_SRCS_DEFAULT += myexampleMain.cpp
myexample_SRCS_vxWorks += -nil-

# Add locally compiled object code
myexample_SRCS += dbSubExample.c

# Add support from base/src/vxWorks if needed
myexample_OBJS_vxWorks += $(EPICS_BASE_BIN)/vxComLibrary

myexample_LIBS += myexampleSupport
myexample_LIBS += $(EPICS_BASE_IOC_LIBS)
```

`PROD_IOC` sets the name of the ioc application, here called `myexample`.

The DBD definition `myexample.dbd` will cause build rules to create the database definition include file `myexampleInclude.dbd` from files in the `myexample_DBD` definition. For each filename in that definition, the created `myexampleInclude.dbd` will contain an include statement for that filename. In this case the created `myexampleInclude.dbd` file will contain the following lines.

```
include "base.dbd"
include "xxxSupport.dbd"
include "dbSubExample.dbd"
```

When the DBD build rules find the created file `myexampleInclude.dbd`, the rules then call dbExpand which reads `myexampleInclude.dbd` to generate file `myexample.dbd`, and install it into `<top>/dbd`.

An arbitrary number of `myexample_SRCS` statements can be given. Names of the form `<name>_registerRecordDeviceDriver.cpp`, are special; when they are seen the perl script `registerRecordDeviceDriver.pl` is executed and given `<name>.dbd` as input. This script generates the `<name>_registerRecordDeviceDriver.cpp` file automatically.

makeBaseApp.pl

`makeBaseApp.pl` is a perl script that creates application areas. It can create the following:

- `<top>/Makefile`
- `<top>/configure` – This directory contains the files needed by the EPICS build system.
- `<top>/xxxApp` – A set of directories and associated files for a major sub-module.
- `<top>/iocBoot` – A subdirectory and associated files.
- `<top>/iocBoot/iocxxx` – A subdirectory and files for a single ioc.

`makeBaseApp.pl` creates directories and then copies template files into the newly created directories while expanding macros in the template files. EPICS base provides two sets of template files: simple and example. These are meant for simple applications. Each site, however, can create its own set of template files which may provide additional functionality. This section describes the functionality of makeBaseApp itself, the next section provides details about the simple and example templates.

Usage

makeBaseApp has four possible forms of command line:

```
<base>/bin/<arch>/makeBaseApp.pl -h
```

Provides help.

```
<base>/bin/<arch>/makeBaseApp.pl -l [options]
```

List the application templates available. This invocation does not alter the current directory.

```
<base>/bin/<arch>/makeBaseApp.pl [-t type] [options] app ...
```

Create application directories.

```
<base>/bin/<arch>/makeBaseApp.pl -i -t type [options] ioc ...
```

Create ioc boot directories.

Options for all command forms:

-b base
: Provides the full path to EPICS base. If not specified, the value is taken from the EPICS_BASE entry in config/RELEASE. If the config directory does not exist, the path is taken from the command-line that was used to invoke makeBaseApp

-T template
: Set the template top directory (where the application templates are). If not specified, the template path is taken from the TEMPLATE_TOP entry in config/RELEASE. If the config directory does not exist the path is taken from the environment variable EPICS_MBA_TEMPLATE_TOP, or if this is not set the templates from EPICS base are used.

-d
: Verbose output (useful for debugging)

Arguments unique to `makeBaseApp.pl [-t type] [options] app ...`:

app
: One or more application names (the created directories will have "App" appended to this name)

-t type
Set the template type (use the -l invocation to get a list of valid types). If this option is not used, type is taken from the environment variable EPICS_MBA_DEF_APP_TYPE, or if that is not set the values "default" and then "example" are tried.

Arguments unique to makeBaseApp.pl -i [options] ioc ...:

ioc
One or more IOC names (the created directories will have "ioc" prepended to this name).

-a arch
Set the IOC architecture (e.g. vxWorks-68040). If -a arch is not specified, you will be prompted.

Environment Variables:

EPICS_MBA_DEF_APP_TYPE
Application type you want to use as default

EPICS_MBA_TEMPLATE_TOP
Template top directory

Description

To create a new <top> issue the commands:

```
mkdir <top>
cd <top>
<base>/bin/<arch>/makeBaseApp.pl -t <type> <app> ...
<base>/bin/<arch>/makeBaseApp.pl -i -t <type> <ioc> ...
```

makeBaseApp does the following:

- EPICS_BASE is located by checking the following in order:
  - If the -b option is specified its value is used.
  - If a <top>/configure/RELEASE file exists and defines a value for EPICS_BASE it is used.
  - It is obtained from the invocation of the makeBaseApp program. For this to work, the full path name to the makeBaseApp.pl script in the EPICS base release you are using must be given.
- TEMPLATE_TOP is located in a similar fashion:
  - If the -T option is specified its value is used.
  - If a <top>/configure/RELEASE file exists and defines a value for TEMPLATE_TOP it is used.
  - If EPICS_MBA_TEMPLATE_TOP is defined its value is used.
  - It is set equal to <epics_base>/templates/makeBaseApp/top
- If -l is specified the list of application types is listed and makeBaseApp terminates.
- If -i is specified and -a is not then the user is prompted for the IOC architecture.
- The application type is determined by checking the following in order:
  - If -t is specified it is used.
  - If EPICS_MBA_DEF_APP_TYPE is defined its value is used.
  - If a template defaultApp exists, the application type is set equal to default.

  - If a template `exampleApp` exists, the application type is set equal to example.
- If the application type is not found in `TEMPLATE_TOP`, makeBaseApp issues an error and terminates.
- If `Makefile` does not exist, it is created.
- If directory `configure` does not exist, it is created and populated with all the `configure` files.
- If `-i` is specified:
  - If directory `iocBoot` does not exist, it is created and the files from the template boot directory are copied into it.
  - For each <ioc> specified on the command line a directory `iocBoot/ioc<ioc>` is created and populated with the files from the template (with `ReplaceLine()` tag replacement, see below).
- If `-i` is NOT specified:
  - For each <app> specified on the command line a directory <app>`App` is created and populated with the directory tree from the template (with `ReplaceLine()` tag replacement, see below).

Tag Replacement within a Template

When copying certain files from the template to the new application structure, makeBaseApp replaces some predefined tags in the name or text of the files concerned with values that are known at the time. An application template can extend this functionality as follows:

Two perl subroutines are defined within makeBaseApp:

`ReplaceFilename`
This substitutes for the following in names of any file taken from the templates.

```
_APPNAME_
_APPTYPE_
```

`ReplaceLine`
This substitutes for the following in each line of each file taken from the templates:

```
_USER_
_EPICS_BASE_
_ARCH_
_APPNAME_
_APPTYPE_
_TEMPLATE_TOP_
_IOC_
```

If the application type directory has a file named `Replace.pl`, this file may:

- Replace one or both of the above subroutines with its own versions.
- Provide a subroutine `ReplaceFilenameHook($file)` which will be called at the end of the subroutine `ReplaceFilename` described above.
- Provide a subroutine `ReplaceLineHook($line)` which is called at the end of `ReplaceLine`.
- Include other code which is run after the command line options have been interpreted.

makeBaseApp templetes provided with base

support

This creates files appropriate for building a support application.

ioc

Without the `-i` option, this creates files appropriate for building an ioc application. With the `-i` option it creates an ioc boot directory.

example

Without the `-i` option it creates files for running an example. Both a support and an ioc application are built. With the `-i` option it creates an ioc boot directory that can be used to run the example.

caClient

This builds two Channel Access clients.

caServer

This builds an example Portable Access Server.

vxWorks boot parameters

The vxWorks boot parameters are set via the console serial port on your IOC. Life is much easier if you can connect the console to a terminal window on your workstation. On Linux the 'screen' program lets you communicate through a local serial port; run `screen /dev/ttyS0` if the IOC is connected to `ttyS0`.

The vxWorks boot parameters look something like the following:

```
boot device            : xxx
processor number       : 0
host name              : xxx
file name              : <full path to board support>/vxWorks
inet on ethernet (e)   : xxx.xxx.xxx.xxx:<netmask>
host inet (h)          : xxx.xxx.xxx.xxx
user (u)               : xxx
ftp password (pw)      : xxx
flags (f)              : 0x0
target name (tn)       : <hostname for this inet address>
startup script (s)     : <top>/iocBoot/iocmyexample/st.cmd
```

The actual values for each field are site and IOC dependent. Two fields that you can change at will are the vxWorks boot image and the location of the startup script.

Note that the full path name for the correct board support boot image must be specified. If bootp is used the same information will need to be placed in the bootp host's configuration database instead.

When your boot parameters are set properly, just press the reset button on your IOC, or use the @ command to commence booting. You will find it VERY convenient to have the console port of the IOC attached to a scrolling window on your workstation.

RTEMS boot procedure

RTEMS uses the vendor-supplied bootstrap mechanism so the method for booting an IOC depends upon the hardware in use.

Booting from a BOOTP/DHCP/TFTP server

Many boards can use BOOTP/DHCP to read their network configuration and then use TFTP to read the applicaion program. RTEMS can then use TFTP or NFS to read startup scripts and configuration files. If you are using TFTP to read the startup scripts and configuration files you must install the EPICS application files on your TFTP server as follows:

- Copy all `db/xxx` files to `<tftpbase>/epics/<target_hostname\>/db/xxx`.
- Copy all `dbd/xxx` files to `<tftpbase>/epics/<target_hostname>/dbd/xxx`.
- Copy the `st.cmd` script to `<tftpbase>/epics/<target_hostname>/st.cmd`.

Use DHCP site-specific option 129 to specify the path to the IOC startup script.

Motorola PPCBUG boot parameters

Motorola single-board computers which employ PPCBUG should have their 'NIOT' parameters set up like:

`Controller LUN =00`
`Device LUN =00`
`Node Control Memory Address =FFE10000`
`Client IP Address` ='Dotted-decimal' IP address of IOC
`Server IP Address` ='Dotted-decimal' IP address of TFTP/NFS server
`Subnet IP Address Mask` ='Dotted-decimal' IP address of subnet mask (255.255.255.0 for class C subnet)
`Broadcast IP Address` ='Dotted-decimal' IP address of subnet broadcast address
`Gateway IP Address` ='Dotted-decimal' IP address of network gateway (0.0.0.0 if none)
`Boot File Name` =Path to application bootable image (..../bin/RTEMS-mvme2100/test.boot)
`Argument File Name` =Path to application startup script (..../iocBoot/ioctest/st.cmd)
`Boot File Load Address =001F0000` (actual value depends on BSP)
`Boot File Execution Address =001F0000` (actual value depends on BSP)
`Boot File Execution Delay =00000000`
`Boot File Length =00000000`
`Boot File Byte Offset =00000000`
`BOOTP/RARP Request Retry =00`
`TFTP/ARP Request Retry =00`
`Trace Character Buffer Address =00000000`

Motorola MOTLOAD boot parameters

Motrola single-board computers which employ MOTLOAD should have their network 'Global Environment Variable' parameters set up like:

`mot-/dev/enet0-cipa`='Dotted-decimal' IP address of IOC
`mot-/dev/enet0-sipa`='Dotted-decimal' IP address of TFTP/NFS server
`mot-/dev/enet0-snma`='Dotted-decimal' IP address of subnet mask (255.255.255.0 for class C subnet)
`mot-/dev/enet0-gipa`='Dotted-decimal' IP address of network gateway (omit if none)
`mot-/dev/enet0-file`=Path to application bootable image (..../bin/RTEMS-mvme5500/test.boot)
`rtems-client-name`=IOC name (mot-/dev/enet0-cipa will be used if this parameter is missing)
`rtems-dns-server`='Dotted-decimal' IP address of domain name server (omit if none)
`rtems-dns-domainname`=Domain name (if this parameter is omitted the compiled-in value will be used)
`epics-script`=Path to application startup script (..../iocBoot/ioctest/st.cmd)

The `mot-script-boot` parameter should be set up like:

```
tftpGet -a4000000 -cxxx -sxxx -mxxx -gxxx -d/dev/enet0
        -f..../bin/RTEMS-mvme5500/test.boot
netShut
go -a4000000
```

where the `-c`, `-s`, `-m` and `-g` values should match the cipa, sipa, snma and gipa values, respectively and the `-f` value should match the file value.

RTEMS NFS access

For IOCs which use NFS for remote file access the EPICS initialization code uses the startup script pathname to determine the parameters for the initial NFS mount. If the startup script pathname begins with a '/' the first component of the pathname is used as both the server path and the local mount point. If the startup script pathname does not begin with a '/' the first component of the pathname is used as the local mount point and the server path is "`/tftpboot/`" followed by the first component of the pathname. This allows the NFS client used for EPICS file access and the TFTP client used for bootstrapping the application to have a similar view of the remote filesystem.

RTEMS 'Cexp'

The RTEMS 'Cexp' add-on package provides the ability to load object modules at application run-time. If your RTEMS build includes this package you can load RTEMS IOC applications in the same fashion as vxWorks IOC applications.

1.13.2 Build Facility

Tags: developer

Janet Anderson is the author of this chapter.

Overview

This chapter describes the EPICS build facility including directory structure, environment and system requirements, configuration files, Makefiles, and related build tools.

<top> Directory structure

EPICS software can be divided into multiple <top> areas. Examples of <top> areas are EPICS base itself, EPICS extensions, and simple or complicated IOC applications. Each <top> may be maintained separately. Different <top> areas can be on different releases of external software such as EPICS base releases.

A <top> directory has the following directory structure:

```
<top>/
├── Makefile
├── configure/
├── dir1/
├── dir2/
├── ...
```

where configure is a directory containing build configuration files and a Makefile, where dir1, dir2, ... are user created subdirectory trees with Makefiles and source files to be built. Because the build rules allow make commands like `make install.vxWorks-68040`, subdirectory names within a <top> directory structure may not contain a period "." character.

Install Directories

Files installed during the build are installed into subdirectories of an installation directory which defaults to `$(TOP)`, the <top> directory. For base, extensions, and IOC applications, the default value can be changed in the `configure/CONFIG_SITE` file. The installation directory for the EPICS components is controlled by the definition of `INSTALL_LOCATION`

The following subdirectories may exist in the installation directory. They are created by the build and contain the installed build components.

- dbd - Directory into which Database Definition files are installed.
- include - The directory into which C header files are installed. These header files may be generated from menu and record type definitions.
- bin - This directory contains a subdirectory for each host architecture and for each target architecture. These are the directories into which executables, binaries, etc. are installed.

- lib - This directory contains a subdirectory for each host architecture. These are the directories into which libraries are installed.
- db - This is the directory into which database record instance, template, and substitution files are installed.
- html - This is the directory into which html documentation is installed.
- templates - This is the directory into which template files are installed.
- javalib - This is the directory into which java class files and jar files are installed.
- configure - The directory into which configure files are installed (if `INSTALL_LOCATION` does not equal `TOP`).
- cfg - The directory into which user created configure files are installed

Elements of build system

The main ingredients of the build system are:

- A set of configuration files and tools provided in the EPICS base/configure directory
- A corresponding set of configuration files in the <top>`/configure` directory of a non-base <top> directory structure to be built. The `makeBaseApp.pl` and `makeBaseExt.pl` scripts create these configuration files. Many of these files just include a file of the same name from the base/configure directory.
- Makefiles in each directory of the <top> directory structure to be built
- User created configuration files in build created `$(INSTALL_LOCATION)/cfg` directories.

Features

The principal features of the build system are:

- Requires a single Makefile in each directory of a <top> directory structure
- Supports both host os vendor's native compiler and GNU compiler
- Supports building multiple types of software (libraries, executables, databases, java class files, etc.) stored in a single directory tree.
- Supports building EPICS base, extensions, and IOC applications.
- Supports multiple host and target operating system + architecture combinations.
- Allows builds for all hosts and targets within a single <top> source directory tree.
- Allows sharing of components such as special record/device/drivers across <top> areas.
- gnumake is the only command used to build a <top> area.

Multiple host and target systems

You can build on multiple host systems and for multiple cross target systems using a single EPICS directory structure. The intermediate and binary files generated by the build will be created in separate `O.*` subdirectories and installed into the appropriate separate host or target install directories. EPICS executables and scripts are installed into the `$(INSTALL_LOCATION)/bin/`<arch> directories. Libraries are installed into `$(INSTALL_LOCATION)/lib/`<arch>. The default definition for `$(INSTALL_LOCATION)` is `$(TOP)` which is the root directory in the directory structure. Architecture dependant created files (e.g. object files) are stored in `O.`<arch> source subdirectories, and architecture independent created files are stored in `O.Common` source subdirectories. This allows objects for multiple cross target architectures to be maintained at the same time.

To build EPICS base for a specific host/target combination you must have the proper host/target c/c++ cross compiler and target header files, CROSS_COMPILER_HOST_ARCHS must empty or include the host architecture in its list value, the CROSS_COMPILER_TARGET_ARCHS variable must include the target to be cross-compiled, and the base/configure/os directory must have the appropriate configure files.

Build Requirements

Host Environment Variable

Only one environment variable, EPICS_HOST_ARCH, is required to build EPICS <top> areas. This variable should be set to be your workstation's operating system - architecture combination to use the os vendor's c/c++ compiler for native builds or set to the operating system - architecture - alternate compiler combination to use an alternate compiler for native builds if an alternate compiler is supported on your system. The filenames of the CONFIG.*.Common files in base/configure/os show the currently supported EPICS_HOST_ARCH values. Examples are solaris-sparc, solaris-sparc-gnu, linux-x86, win32-x86, and cygwin-x86.

Software Prerequisites

Before you can build EPICS components your host system must have the following software installed:

- Perl version 5.8 or greater
- GNU make, version 3.81 or greater
- C++ compiler (host operating system vendor's compiler or GNU compiler)

If you will be building EPICS components for vxWorks targets you will also need:

- Tornado II or vxWorks 6.x or later, and one or more board support packages. Consult the vxWorks documentation for details.

If you will be building EPICS components for RTEMS targets you will also need:

- RTEMS development tools and libraries required to run EPICS IOC applications.

Path requirements

You must have the perl executable in your path and you may need C and C++ compilers in your search path. Check definitions of CC and CCC in base/configure/os/CONFIG.<host>.<host> or the definitions for GCC and G++ if ANSI=GCC and CPLUSPLUS=GCC are specified in CONFIG_SITE. For building base you also must have echo in your search path. You can override the default settings by defining PERL, CC and CCC, GCC and G++, GNU_DIR ... in the appropriate file (usually configure/os/CONFIG_SITE.$EPICS_HOST_ARCH.Common)

Unix path

For Unix host builds you also need touch, cpp, cp, rm, mv, and mkdir in your search path and /bin/chmod must exist. On some Unix systems you may also need ar and ranlib in your path, and the c compiler may require ld in your path.

Win32 PATH

On WIN32 systems, building shared libraries is the default setting and you will need to add fullpathname to `$(INSTALL_LOCATION)/bin/$(EPICS_HOST_ARCH)` to your path so the shared libraries, dlls, can be found during the build... Building shared libraries is determined by the value of the macro `SHARED_LIBRARIES` in `CONFIG_SITE` or `os/CONFIG.Common.<host>` (either `YES` or `NO`).

Directory names

Because the build rules allow make commands like `make <dir>.<action>,<arch>`, subdirectory names within a `<top>` directory structure may not contain a period"." character.

`EPICS_HOST_ARCH` environment variable

The startup directory in EPICS base contains a perl script, `EpicsHostArch.pl`, which can be used to define `EPICS_HOST_ARCH`. This script can be invoked with a command line parameter defining the alternate compiler (e.g. if invoking `EpicsHostArch.pl` yields solaris-sparc, then invoking `EpicsHostArch.pl` gnu will yield solaris-sparc-gnu).

The startup directory also contains scripts to help users set the path and other environment variables.

Configuration Definitions

Site-specific EPICS Base Configuration

Site configuration

To configure EPICS base for your site, you may want to modify the default definitions in the following files:

- `configure/CONFIG_SITE` - Build choices. Specify target archs.
- `configure/CONFIG_SITE_ENV` - Environment variable defaults

Host configuration

To configure each host system for your site, you may override the default definitions in the configure/os directory by adding a new file with override definitions. The new file should have the same name as the distribution file to be overridden except `CONFIG` in the name is changed to `CONFIG_SITE`.

- `configure/os/CONFIG_SITE.<host>.<host>` - Host build settings
- `configure/os/CONFIG_SITE.<host>.Common` - Host build settings for all target systems

Target configuration

To configure each target system, you may override the default definitions in the configure/os directory by adding a new file with override definitions. The new file should have the same name as the distribution file to be overridden except `CONFIG` in the name is replaced by `CONFIG_SITE`.

- `configure/os/CONFIG_SITE.Common.<target>` - Target cross settings
- `configure/os/CONFIG_SITE.<host>.<target>` - Host-target settings
- `configure/os/CONFIG_SITE.Common.vxWorksCommon` - vxWorks full paths

R3.13 compatibility configuration

To configure EPICS base for building with R3.13 extensions and ioc applications, you must modify the default definitions in the `base/config/CONFIG_SITE*` files to agree with site definitions you made in base/configure and base/configure/os files.You must also modify the following tow macros in the `base/configure/CONFIG_SITE` file:

- `COMPAT_TOOLS_313` - Set to `YES` to build R3.13 extensions with this base.
- `COMPAT_313` - Set to `YES` to build R3.13 ioc applications and extensions with this base.

Directory definitions

The configure files contain definitions for locations in which to install various components. These are all relative to `INSTALL_LOCATION`. The default value for `INSTALL_LOCATION` is `$(TOP)`, and `$(T_A)` is the current build's target architecture. The default value for `INSTALL_LOCATION` can be overridden in the `configure/CONFIG_SITE` file.

```
INSTALL_LOCATION_LIB = $(INSTALL_LOCATION)/lib
INSTALL_LOCATION_BIN = $(INSTALL_LOCATION)/bin

INSTALL_HOST_BIN = $(INSTALL_LOCATION_BIN)/$(EPICS_HOST_ARCH)
INSTALL_HOST_LIB = $(INSTALL_LOCATION_LIB)/$(EPICS_HOST_ARCH)

INSTALL_INCLUDE = $(INSTALL_LOCATION)/include
INSTALL_DOC = $(INSTALL_LOCATION)/doc
INSTALL_HTML = $(INSTALL_LOCATION)/html
INSTALL_TEMPLATES = $(INSTALL_LOCATION)/templates
INSTALL_DBD = $(INSTALL_LOCATION)/dbd
INSTALL_DB = $(INSTALL_LOCATION)/db
INSTALL_CONFIG = $(INSTALL_LOCATION)/configure
INSTALL_JAVA = $(INSTALL_LOCATION)/javalib

INSTALL_LIB = $(INSTALL_LOCATION_LIB)/$(T_A)
INSTALL_SHRLIB = $(INSTALL_LOCATION_LIB)/$(T_A)
INSTALL_TCLLIB = $(INSTALL_LOCATION_LIB)/$(T_A)
INSTALL_BIN = $(INSTALL_LOCATION_BIN)/$(T_A)
```

Extension and Application Specific Configuration

The base/configure directory contains files with the default build definitions and site specific build definitions. The extensions/configure directory contains extension specific build definitions (e.g. location of X11 and Motif libraries) and `include <filename>` lines for the base/configure files. Likewise, the `<application>/configure` directory contains application specific build definitions and includes for the application source files. Build definitions such as `CROSS_COMPILER_TARGET_ARCHS` can be overridden in an extension or application by placing an override definition in the `<top>/configure/CONFIG_SITE` file.

RELEASE file

Every `<top>/configure` directory contains a `RELEASE` file. `RELEASE` contains a user specified list of other `<top>` directory structures containing files needed by the current `<top>`, and may also include other files to take those definitions from elsewhere. The macros defined in the `RELEASE` file (or its includes) may reference other defined macros, but cannot rely on environment variables to provide definitions.

When make is executed, macro definitions for include, bin, and library directories are automatically generated for each external `<top>` definition given in the `RELEASE` file. Also generated are include statements for any existing `RULES_BUILD` files, `cfg/RULES*` files, and `cfg/CONFIG*` files from each external `<top>` listed in the `RELEASE` file.

For example, if `configure/RELEASE` contains the definition

```
CAMAC = /home/epics/modules/bus/camac
```

then the generated macros will be:

```
CAMAC_HOST_BIN = /home/epics/modules/bus/camac/bin/$(EPICS_HOST_ARCH)
CAMAC_HOST_LIB = /home/epics/modules/bus/camac/lib/$(EPICS_HOST_ARCH)
CAMAC_BIN = /home/epics/modules/bus/camac/bin/$(T_A)
CAMAC_LIB = /home/epics/modules/bus/camac/lib/$(T_A)
RELEASE_INCLUDES += -I/home/epics/modules/bus/camac/include/os
RELEASE_INCLUDES += -I/home/epics/modules/bus/camac/include
RELEASE_DBDFLAGS += -I /home/epics/modules/bus/camac/dbd
RELEASE_DBFLAGS += -I/home/epics/modules/bus/camac/db
RELEASE_PERL_MODULE_DIRS += /home/epics/modules/bus/camac/lib/perl
```

`RELEASE_DBDFLAGS` will appear on the command lines for the `dbToRecordTypeH`, `mkmf.pl`, and `dbExpand` tools, and `RELEASE_INCLUDES` will appear on compiler command lines. `CAMAC_LIB` and `CAMAC_BIN` can be used in a Makefile to define the location of needed scripts, executables, object files, libraries or other files.

Definitions in `configure/RELEASE` can be overridden for a specific host and target architectures by providing the appropriate file or files containing overriding definitions.

- `configure/RELEASE.<epics_host_arch>.Common`
- `configure/RELEASE.Common.<targetarch>`
- `configure/RELEASE.<epics_host_arch>.<targetarch>`

For `<top>` directory structures created by `makeBaseApp.pl`, an EPICS base perl script, `convertRelease.pl` can perform consistency checks for the external `<top>` definitions in the `RELEASE` file and its includes as part of the `<top>` level build. Consistancy checks are controlled by value of `CHECK_RELEASE` which is defined in `<top>/configure/CONFIG_SITE`. `CHECK_RELEASE` can be set to `YES`, `NO` or `WARN`, and if `YES` (the default value), consistency checks will be performed. If `CHECK_RELEASE` is set to `WARN` the build will continue even if conflicts are found.

Modifying configure/RELEASE* files

You should always do a gnumake clean uninstall in the <top> directory BEFORE adding, changing, or removing any definitions in the configure/RELEASE* files and then a gnumake at the top level AFTER making the changes.

The file <top>/configure/RELEASE contains definitions for components obtained from outside <top>. If you want to link to a new release of anything defined in the file do the following:

```
$ cd <top>
$ make clean uninstall
```

edit configure/RELEASE, and change the relevant line(s) to point to the new release

```
$ make
```

All definitions in <top>/configure/RELEASE must result in complete path definitions, i.e. relative path names are not permitted. If your site could have multiple releases of base and other support <top> components installed at once, these path definitions should contain a release number as one of the components. However as the RELEASE file is read by gnumake, it is permissible to use macro substitutions to define these pathnames, for example:

```
SUPPORT = /usr/local/iocapps/R3.14.9
EPICS_BASE = $(SUPPORT)/base/3-14-9-asd1
```

OS Class specific definitions

Definitions in a Makefile will apply to the host system (the platform on which make is executed) and each system defined by CROSS_COMPILER_TARGET_ARCHS.

It is possible to limit the architectures for which a particular definition is used. Most Makefile definition names can be specified with an appended underscore "_" followed by an osclass name. If an _<osclass> is not specified, then the definition applies to the host and all CROSS_COMPILER_TARGET_ARCHS systems. If an _<osclass> is specified, then the definition applies only to systems with the specified os class. A Makefile definition can also have an appended _DEFAULT specification. If _DEFAULT is appended, then the Makefile definition will apply to all systems that do not have an _<osclass> specification for that definition. If a _DEFAULT definition exists but should not apply to a particular system OS Class, the value -nil- should be specified in the relevant Makefile definition.

Each system has an OS_CLASS definition in its configure/os/CONFIG.Common.<arch> file. A few examples are:

- For vxWorks-* targets OS_CLASS is vxWorks.
- For RTEMS-* targets OS_CLASS is RTEMS.
- For solaris-* targets OS_CLASS is solaris.
- For win32-* targets OS_CLASS is WIN32.
- For linux-* targets OS_CLASS is Linux.
- For darwin-* targets OS_CLASS is Darwin.
- For aix-* targets OS_CLASS is AIX.

For example the following Makefile lines specify that product aaa should be created for all systems. Product bbb should be created for systems that do not have OS_CLASS defined as solaris.

```
PROD = aaa
PROD_solaris = -nil-
PROD_DEFAULT = bbb
```

Specifying T_A specific definitions

It is possible for the user to limit the systems for which a particular definition applies to specific target systems.

For example the following Makefile lines specify that product aaa should be created for all target architecture which allow IOC type products and product bbb should be created only for the vxWorks-68040 and vxWorks-ppc603 targets. Remember T_A is the build's current target architecture. so PROD_IOC has the bbb value only when the current built target architecture is vwWorks-68040 or vxWorks-ppc603

```
PROD_IOC = aaa
VX_PROD_vxWorks-68040 = bbb
VX_PROD_vxWorks-ppc603 = bbb
PROD_IOC += VX_PROD_$(T_A)
```

Host and Ioc targets

Build creates two type of makefile targets: Host and Ioc. Host targets are executables, object files, libraries, and scripts which are not part of iocCore. Ioc targets are components of ioc libraries, executables, object files, or iocsh scripts which will be run on an ioc.

Each supported target system has a VALID_BUILDS definition which specifies the type of makefile targets it can support. This definition appears in configure/os/CONFIG.Common.<arch> or configure/os/CONFIG.<arch>.<arch> files.

- For vxWorks systems VALID_BUILDS is set to "Ioc".
- For Unix type systems, VALID_BUILDS is set to "Host Ioc".
- For RTEMS systems, VALID_BUILDS is set to "Ioc".
- For WIN32 systems, VALID_BUILDS is set to "Host Ioc".

In a Makefile it is possible to limit the systems for which a particular PROD, TESTPROD, LIBRARY, SCRIPTS, and OBJS is built. For example the following Makefile lines specify that product aaa should be created for systems that support Host type builds. Product bbb should be created for systems that support Ioc type builds. Product ccc should be created for all target systems.

```
PROD_HOST = aaa
PROD_IOC = bbb
PROD = ccc
```

These definitions can be further limited by specifying an appended underscore "_" followed by an osclass or DEFAULT specification.

User specific override definitions

User specific override definitions are allowed in user created files in the user's <home>/configure subdirectory. These override definitions will be used for builds in all <top> directory structures. The files must have the following names.

- <home>/configure/CONFIG_USER
- <home>/configure/CONFIG_USER.<epics_host_arch>
- <home>/configure/CONFIG_USER.Common.<targetarch>
- <home>/configure/CONFIG_USER.<epics_host_arch>.<targetarch>

Makefiles

Name

The name of the makefile in each directory must be Makefile.

Included Files

Makefiles normally include files from <top>/configure. Thus the makefile "inherits" rules and definitions from configure. The files in <top>/configure may in turn include files from another <top>/configure. This technique makes it possible to share make variables and even rules across <top> directories.

Contents of Makefiles

Makefiles in directories containing subdirectories

A Makefile in this type of directory must define where <top> is relative to this directory, include <top>/configure files, and specify the subdirectories in the desired order of make execution. Running gnumake in a directory with the following Makefile lines will cause gnumake to be executed in first and then . The build rules do not allow a Makefile to specify both subdirectories and components to be built.

```
TOP=../..
include $(TOP)/configure/CONFIG
DIRS += <dir1> <dir2>
include $(TOP)/configure/RULES_DIRS
```

Makefiles in directories where components are to be built

A Makefile in this type of directory must define where <top> is relative to this directory, include <top> configure files, and specify the target component definitions. Optionally it may contain user defined rules. Running gnumake in a directory with this type of Makefile will cause gnumake to create an O.<arch> subdirectory and then execute gnumake to build the defined components in this subdirectory. It contains the following lines:

```
TOP=../../..
include $(TOP)/configure/CONFIG
# <component definition lines>
include $(TOP)/configure/RULES
# <optional rules definitions>
```

Simple Makefile examples

Create an IOC type library named asIoc from the source file asDbLib.c and install it into the $(INSTALL_LOCATION)/lib/<arch> directory.

```
TOP=../../..
include $(TOP)/configure/CONFIG
LIBRARY_IOC += asIoc
asIoc_SRCS += asDbLib.c
include $(TOP)/configure/RULES
```

For each Host type target architecture, create an executable named catest from the `catest1.c` and `catest2.c` source files linking with the existing EPICS base ca and Com libraries, and then install the catest executable into the `$(INSTALL_LOCATION)/bin/<arch>` directory.

```
TOP=../../..
include $(TOP)/configure/CONFIG
PROD_HOST = catest
catest_SRCS += catest1.c catest2.c
catest_LIBS = ca Com
include $(TOP)/configure/RULES
```

Make

Make vs. gnumake

EPICS provides an extensive set of make rules. These rules only work with the GNU version of make, gnumake, which is supplied by the Free Software Foundation. Thus, on most Unix systems, the native make will not work. On some systems, e.g. Linux, GNU make may be the default. This manual always uses gnumake in the examples.

Frequently used Make commands

NOTE

It is possible to invoke the following commands for a single target architecture by appending `<arch>` to the target in the command.

The most frequently used make commands are:

gnumake
: This rebuilds and installs everything that is not up to date.

  NOTE

  Executing gnumake without arguments is the same as `gnumake install`

  ---

gnumake help
: This command can be executed from the `<top>` directory only. This command prints a page describing the most frequently used make commands.

gnumake install
: This rebuilds and installs everything that is not up to date.

gnumake all
: This is the same as `gnumake install`.

gnumake buildInstall
: This is the same as `gnumake install`.

gnumake
: This rebuilds and installs everything that is not up to date first for the host arch and then (if different) for the specified target arch.

NOTE

This is the same as `gnumake install.<arch>`

gnumake clean
This can be used to save disk space by deleting the `O.<arch>` directories that gnumake will create, but does not remove any installed files from the bin, db, dbd etc. directories. `gnumake clean.<arch>` can be invoked to clean a single architecture.

gnumake archclean
This command will remove the current build's `O.<arch>` directories but not `O.Common` directory.

gnumake realclean
This command will remove ALL the `O.<arch>` subdirectories (even those created by a gnumake from another `EPICS_HOST_ARCH`).

gnumake rebuild
This is the same as `gnumake clean install`. If you are unsure about the state of the generated files in an application, just execute `gnumake rebuild`.

gnumake uninstall
This command can be executed from the `<top>` directory only. It will remove everything installed by gnumake in the include, lib, bin, db, dbd, etc. directories.

gnumake realuninstall
This command can be executed from the `<top>` directory only. It will remove all the install directories, include, lib, bin, db, dbd, etc.

gnumake distclean
This command can be executed from the `<top>` directory only. It is the same as issuing both the realclean and realuninstall commands.

gnumake cvsclean
This command can be executed from the `<top>` directory only. It removes cvs .#* files in the make directory tree.

Make targets

The following is a summary of targets that can be specified for gnumake:

- `<action>`
- `<arch>`
- `<action>.<arch>`
- `<dir>`
- `<dir>.<action>`
- `<dir>.<arch>`
- `<dir>.<action>.<arch>`

where:

- `<action>` is help, clean, realclean, distclean, inc, install, build, rebuild, buildInstall, realuninstall, or uninstall
- `<arch>` is an architecture such as solaris-sparc, vxWorks-68040, win32-x86, etc.
- `<dir>` is a path in module directory.

NOTE

help, uninstall, distclean, cvsclean, and realuninstall can only be specified at <top>.

realclean cannot be specified inside an O.<arch> subdirectory. <dir> is subdirectory name

NOTE

You can build using your os vendor's native compiler and also build using a supported alternate compiler in the same directory structure because the executables and libraries will be created and installed into separate directories (e.g bin/solaris-sparc and bin/solaris-sparc-gnu). You can do this by changing your EPICS_HOST_ARCH, environment variable between builds or by setting EPICS_HOST_ARCH on the gnumake command line.

The build system ensures the host architecture is up to date before building a cross-compiled target, thus Makefiles must be explicit in defining which architectures a component should be built for.

Header file dependencies

All product, test product, and library source files which appear in one of the source file definitions (e.g. SRCS, PROD_SRCS, LIB_SRCS, <prodname>_SRCS) will have their header file dependencies automatically generated and included as part of the Makefile.

Makefile definitions

The following components can be defined in a Makefile:

Source file directories

Normally all product, test product, and library source files reside in the same directory as the Makefile. OS specific source files are allowed and should reside in subdirectories os/<os_class> or os/posix or os/default.

The build rules also allow source files to reside in subdirectories of the current Makefile directory (src directory). For each subdirectory <dir> containing source files add the SRC_DIRS definition.

```
SRC_DIRS += <dir>
```

where <dir> is a relative path definition. An example of SRC_DIRS is

```
SRC_DIRS += ../dir1 ../dir2
```

The directory search order for the above definition is

```
. ../os/$(OS_CLASS) ../os/posix ../os/default ../dir1/os/$(OS_CLASS)
../dir1/os/posix ../dir1/os/default ../dir2/os/$(OS_CLASS)
../dir2/os/posix ../dir2/os/default .. ../dir1 ../dir2
```

where the build directory O.<arch> is . and the src directory is ...

Posix C source code

The epics base config files assume posix source code and define POSIX to be YES as the default. Individual Makefiles can override this by setting POSIX to NO. Source code files may have the suffix .c, .cc, .cpp, or .C.

Breakpoint Tables

For each breakpoint table dbd file, bpt<table name>.dbd, to be created from an existing bpt<table name>.data file, add the definition

```
DBD += bpt<table name>.dbd
```

to the Makefile. The following Makefile will create a bptTypeJdegC.dbd file from an existing bptTypeJdegC.data file using the EPICS base utility program makeBpt and install the new dbd file into the $(INSTALL_LOCATION)/dbd directory.

```
TOP=../../..
include $(TOP)/configure/CONFIG
DBD += bptTypeJdegC.dbd
include $(TOP)/configure/RULES
```

Record Type Definitions

For each new record type, the following definition should be added to the makefile:

```
DBDINC += <rectype>Record
```

A <rectype>Record.h header file will be created from an existing <rectype>Record.dbd file using the EPICS base utility program dbToRecordTypeH. This header will be installed into the $(INSTALL_LOCATION)/include directory and the dbd file will be installed into the $(INSTALL_LOCATION)/dbd directory.

The following Makefile will create xxxRecord.h from an existing xxxRecord.dbd file, install xxxRecord.h into $(INSTALL_LOCATION)/include, and install xxxRecord.dbd into $(INSTALL_LOCATION)/dbd.

```
TOP=../../..
include $(TOP)/configure/CONFIG
DBDINC += xxxRecord
include $(TOP)/configure/RULES
```

Menus

If a menu menu<name>.dbd file is present, then add the following definition:

```
DBDINC += menu<name>.h
```

The header file, menu<name>.h will be created from the existing menu<name>.dbd file using the EPICS base utility program dbToMenuH and installed into the $(INSTALL_LOCATION)/include directory and the menu dbd file will be installed into $(INSTALL_LOCATION)/dbd.

The following Makefile will create a menuConvert.h file from an existing menuConvert.dbd file and install menuConvert.h into $(INSTALL_LOCATION)/include and menuConvert.dbd into $(INSTALL_LOCATION)/dbd.

```
TOP=../../..
include $(TOP)/configure/CONFIG
DBDINC = menuConvert.h
include $(TOP)/configure/RULES
```

Expanded Database Definition Files

Database definition include files named <name>Include.dbd containing includes for other database definition files can be expanded by the EPICS base utility program dbExpand into a created <name>.dbd file and the <name>.dbd file installed into $(INSTALL_LOCATION)/dbd. The following variables control the process:

```
DBD += <name>.dbd
USR_DBDFLAGS += -I <include path>
USR_DBDFLAGS += -S <macro substitutions>
<name>_DBD += <file1>.dbd <file2>.dbd ...
```

where

```
DBD += <name>.dbd
```

is the name of the output dbd file to contain the expanded definitions. It is created by expanding an existing or build created <name>Include.dbd file and then copied into $(INSTALL_LOCATION)/dbd.

An example of a file to be expanded is exampleInclude.dbd containing the following lines

```
include "base.dbd"
include "xxxRecord.dbd"
device(xxx,CONSTANT,devXxxSoft,"SoftChannel")
```

USR_DBDFLAGS defines optional flags for dbExpand. Currently only an include path (-I <path>) and macro substitution (-S <substitution>) are supported. The include paths for EPICS base/dbd, and other <top>/dbd directories will automatically be added during the build if the <top> names are specified in the configure/RELEASE file.

A database definition include file named <name>Include.dbd containing includes for other database definition files can be created from a <name>_DBD definition. The lines

```
DBD += <name>.dbd
<name>_DBD += <file1>.dbd <file2>.dbd ...
```

will create an expanded dbd file <name>.dbd by first creating a <name>Include.dbd. For each filename in the <name>_DBD definition, the created <name>Include.dbd will contain an include statement for that filename. Then the expanded DBD file is generated from the created <name>Include.dbd file and installed into $(INSTALL_LOCATION)/dbd.

The following Makefile will create an expanded dbd file named example.dbd from an existing exampleInclude.dbd file and then install example.dbd into the $(INSTALL_LOCATION)/dbd directory.

```
TOP=../../..
include $(TOP)/configure/CONFIG
DBD += exampleApp.dbd
include $(TOP)/configure/RULES
```

The following Makefile will create an exampleInclude.dbd file from the example_DBD definition then expand it to create an expanded dbd file, example.dbd, and install example.dbd into the $(INSTALL_LOCATION)/dbd directory.

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

The created `exampleInclude.dbd` file will contain the following lines

```
include "base.dbd"
include "xxxRecord.dbd"
include "xxxSupport.dbd"
```

Registering Support Routines for Expanded Database Definition Files

A source file which registers simple static variables and record/device/driver support routines with iocsh can be created. The list of variables and routines to register is obtained from lines in an existing dbd file.

The following line in a Makefile will result in `<name>_registerRecordDeviceDriver.cpp` being created, compiled, and linked into `<prodname>`. It requires that the file `<name>.dbd` exist or can be created using other make rules.

```
<prodname>_SRCS += <name>_registerRecordDeviceDriver.cpp
```

An example of registering the variable `mySubDebug` and the routines `mySubInit` and `mySubProcess` is `<name>.dbd` containg the following lines

```
variable(mySubDebug)
function(mySubInit)
function(mySubProcess)
```

Database Definition Files

The following line installs the existing named dbd files into `$(INSTALL_LOCATION)/dbd` without expansion.

```
DBD += <name>.dbd
```

DBD install files

Definitions of the form:

```
DBD_INSTALLS += <name>
```

result in files being installed to the `$(INSTALL_LOCATION)/dbd` directory. The file `<name>` can appear with or without a directory prefix. If the file has a directory prefix e.g. `$(APPNAME)/dbd/`, it is copied from the specified location. If a directory prefix is not present, make will look in the current source directory for the file.

Database Files

For most databases just the name of the database has to be specified. Make will figure out how to generate the file:

```
DB += xxx.db
```

generates xxx.db depending on which source files exist and installs it into $(INSTALL_LOCATION)/db.

A <name>.db database file will be created from an optional <name>.template file and/or an optional <name>.substitutions file, If the substitution file exists but the template file is not named <name>.template, the template file name can be specified as

```
<name>_TEMPLATE = <template file name>
```

A *<nn>.db database file will be created from a *.template and a *<nn>.substitutions file, (where nn is an optional index number).

If a <name> substitutions file contains "file" references to other input files, these referenced files are made dependencies of the created <name>.db by the makeDbDepends.pl perl tool.

The Macro Substitutions and Include tool, msi, will be used to generate the database, and msi must either be in your path or you must redefine MSI as the full path name to the msi binary in a RELEASE file or Makefile. An example MSI definition is

```
MSI = /usr/local/epics/extensions/bin/${EPICS_HOST_ARCH}/msi
```

Template files <name>.template, and db files, <name>.db, will be created from an edf file <name>.edf and an <name>.edf file will be created from a <name>.sch file.

Template and substitution files can be installed.

```
DB += xxx.template xxx.substitutions
```

generates and installs these files. If one or more xxx.substitutions files are to be created by script, the script name must be placed in the CREATESUBSTITUTIONS variable (e.g. CREATESUBSTITUTIONS=mySubst.pl). This script will be executed by gnumake with the prefix of the substitution file name to be generated as its argument. If (and only if) there are script generated substitutions files, the prefix of any inflated database's name may not equal the prefix of the name of any template used within the directory.

DB install files

Definitions of the form:

```
DB_INSTALLS += <name>
```

result in files being installed to the $(INSTALL_LOCATION)/db directory. The file <name> can appear with or without a directory prefix. If the file has a directory prefix e.g. $(APPNAME)/db/, it is copied from the specified location. If a directory prefix is not present, make will look in the current source directory for the file.

Compile and link command options

Any of the following can be specified:

Options for all compile/link commands.

These definitions will apply to all compiler and linker targets.

```
USR_INCLUDES += -I<name>
```

header file directories each prefixed by a `-I`.

```
USR_INCLUDES_<osclass> += -I<name>
```

os specific header file directories each prefixed by a `-I`.

```
USR_INCLUDES_DEFAULT += -I<name>
```

header file directories each prefixed by `-I` for any arch that does not have a `USR_INCLUDE_<osclass>` definition

```
USR_CFLAGS += <c flags>
```

C compiler options.

```
USR_CFLAGS_<osclass> += <c flags>
```

os specific C compiler options.

```
USR_CFLAGS_<arch> += <c flags>
```

target architecture specific C compiler options.

```
USR_CFLAGS_DEFAULT += <c flags>
```

C compiler options for any arch that does not have a `USR_CFLAGS_<osclass>` definition

```
USR_CXXFLAGS += <c++ flags>
```

C++ compiler options.

```
USR_CXXFLAGS_<osclass> += <c++ flags>
```

C++ compiler options for the specified osclass.

```
USR_CXXFLAGS_<arch> += <c++ flags>
```

C++ compiler options for the specified target architecture.

```
USR_CXXFLAGS_DEFAULT += <c++ flags>
```

C++ compiler options for any arch that does not have a `USR_CXXFLAGS_<osclass>` definition

```
USR_CPPFLAGS += <preprocessor flags>
```

C preprocessor options.

```
USR_CPPFLAGS_<osclass> += <preprocessor flags>
```

os specific C preprocessor options.

```
USR_CPPFLAGS_<arch> += <preprocessor flags>
```

target architecture specific C preprocessor options.

```
USR_CPPFLAGS_DEFAULT += <preprocessor flags>
```

C preprocessor options for any arch that does not have a USR_CPPFLAGS_<osclass> definition

```
USR_LDFLAGS += <linker flags>
```

linker options.

```
USR_LDFLAGS_<osclass> += <linker flags>
```

os specific linker options.

```
USR_LDFLAGS_DEFAULT += <linker flags>
```

linker options for any arch that does not have a USR_LDFLAGS_<osclass> definition

Options for a target specific compile/link command.

```
<name>_INCLUDES += -I<name>
```

header file directories each prefixed by a -I.

```
<name>_INCLUDES_<osclass> += -I<name>
```

os specific header file directories each prefixed by a -I.

```
<name>_INCLUDES_<T_A> += -I<name>
```

target architecture specific header file directories each prefixed by a -I.

```
<name>_CFLAGS += <c flags>
```

c compiler options.

```
<name>_CFLAGS_<osclass> += <c flags>
```

os specific c compiler options.

```
<name>_CFLAGS_<T_A> += <c flags>
```

target architecture specific c compiler options.

```
<name>_CXXFLAGS += <c++ flags>
```

c++ compiler options.

```
<name>_CXXFLAGS_<osclass> += <c++ flags>
```

c++ compiler options for the specified osclass.

```
<name>_CXXFLAGS_<T_A> += <c++ flags>
```

c++ compiler options for the specified target architecture.

```
<name>_CPPFLAGS += <preprocessor flags>
```

c preprocessor options.

```
<name>_CPPFLAGS_<osclass> += <preprocessor flags>
```

os specific c preprocessor options.

```
<name>_CPPFLAGS_<T_A> += <preprocessor flags>
```

target architecture specific c preprocessor options.

```
<name>_LDFLAGS += <linker flags>
```

linker options.

```
<name>_LDFLAGS_<osclass> += <linker flags>
```

os specific linker options.

Libraries

A library is created and installed into `$(INSTALL_LOCATION)/lib/<arch>` by specifying its name and the name of the object and/or source files containing code for the library. An object or source file name can appear with or without a directory prefix. If the file name has a directory prefix e.g. `$(EPICS_BASE_BIN)`, it is taken from the specified location. If a directory prefix is not present, make will first look in the source directories for a file with the specified name and next try to create the file using existing configure rules. A library filename prefix may be prepended to the library name when the file is created. For Unix type systems and vxWorks the library prefix is lib and there is no prefix for WIN32. Also a library suffix appropriate for the library type and target arch (e.g. .a, .so, .lib, .dll) will be appended to the filename when the file is created.

---

vxWorks and RTEMS

Only archive libraries are created.

Shared libraries

Shared libraries can be built for any or all HOST type architectures. The definition of `SHARED_LIBRARIES` (YES/NO) in `base/configure/CONFIG_SITE` determines whether shared or archive libraries will be built. When `SHARED_LIBRARIES` is `YES`, both archive and shared libraries are built. This definition can be overridden for a specific arch in an `configure/os/CONFIG_SITE.<arch>.Common` file. The default definition for `SHARED_LIBRARIES` in the EPICS base distribution file is `YES` for all host systems.

Win32

An object library file is created when SHARED_LIBRARIES=NO, <name>.lib which is installed into $(INSTALL_LOCATION)/lib/<arch>. Two library files are created when SHARED_LIBRARIES=YES, <name>.lib, an import library for DLLs, which is installed into $(INSTALL_LOCATION)/lib/<arch>, and <name>.dll which is installed into $(INSTALL_LOCATION)/bin/<arch>. (Warning: The file <name>.lib will only be created by the build if there are exported symbols from the library.) If SHARED_LIBRARIES=YES, the directory $(INSTALL_LOCATION)/bin/<arch> must be in the user's path during builds to allow invoking executables which were linked with shared libraries.

Note that the <name>.lib files are different for shared and nonshared builds.

Specifying the library name.

Any of the following can be specified:

```
LIBRARY += <name>
```

A library will be created for every target arch.

```
LIBRARY_<osclass> += <name>
```

Library <name> will be created for all archs of the specified osclass.

```
LIBRARY_DEFAULT += <name>
```

Library <name> will be created for any arch that does not have a LIBRARY_<osclass> definition

```
LIBRARY_IOC += <name>
```

Library <name> will be created for IOC type archs.

```
LIBRARY_IOC_<osclass> += <name>
```

Library <name> will be created for all IOC type archs of the specified osclass.

```
LIBRARY_IOC_DEFAULT += <name>
```

Library <name> will be created for any IOC type arch that does not have a LIBRARY_IOC_<osclass> definition

```
LIBRARY_HOST += <name>
```

Library <name> will be created for HOST type archs.

```
LIBRARY_HOST_<osclass> += <name>
```

Library <name> will be created for all HOST type archs of the specified osclass.

```
LIBRARY_HOST_DEFAULT += <name>
```

Library <name> will be created for any HOST type arch that does not have a LIBRARY_HOST_<osclass> definition

Specifying library source file names

Source file names, which must have a suffix, are defined as follows:

```
SRCS += <name>
```

Source files will be used for all defined libraries and products.

```
SRCS_<osclass> += <name>
```

Source files will be used for all defined libraries and products for all archs of the specified osclass.

```
SRCS_DEFAULT += <name>
```

Source files will be used for all defined libraries and products for any arch that does not have a SRCS_<osclass> definition

LIBSRCS and LIB_SRCS have the same meaning. LIBSRCS is deprecated, but retained for R3.13 compatibility.

```
LIBSRCS += <name>
```

Source files will be used for all defined libraries.

```
LIBSRCS_<osclass> += <name>
```

Source files will be used for all defined libraries for all archs of the specified osclass.

```
LIBSRCS_DEFAULT += <name>
```

Source files will be used for all defined libraries for any arch that does not have a LIBSRCS_<osclass> definition

```
USR_SRCS += <name>
```

Source files will be used for all defined products and libraries.

```
USR_SRCS_<osclass> += <name>
```

Source files will be used for all defined products and libraries for all archs of the specified osclass.

```
USR_SRCS_DEFAULT += <name>
```

Source files will be used for all defined products and libraries for any arch that does not have a USR_SRCS_<osclass> definition

```
LIB_SRCS += <name>
```

Source files will be used for all libraries.

```
LIB_SRCS_<osclass> += <name>
```

Source files will be used for all defined libraries for all archs of the specified osclass.

```
LIB_SRCS_DEFAULT += <name>
```

Source files will be used for all defined libraries for any arch that does not have a LIB_SRCS_<osclass> definition

```
<libname>_SRCS += <name>
```

Source files will be used for the named library.

```
<libname>_SRCS_<osclass> += <name>
```

Source files will be used for named library for all archs of the specified osclass.

```
<libname>_SRCS_DEFAULT += <name>
```

Source files will be used for named library for any arch that does not have a <libname>_SRCS_<osclass> definition

Specifying library object file names

Library object file names should only be specified for object files which will not be built in the current directory. For object files built in the current directory, library source file names should be specified. See Specifying Library Source File Names above.

Object files which have filename with a ".o" or ".obj" suffix are defined as follows and can be specified without the suffix but should have the directory prefix

```
USR_OBJS += <name>
```

Object files will be used in builds of all products and libraries

```
USR_OBJS_<osclass> += <name>
```

Object files will be used in builds of all products and libraries for archs with the specified osclass.

```
USR_OBJS_DEFAULT += <name>
```

Object files will be used in builds of all products and libraries for archs without a USR_OBJS_<osclass> definition specified.

```
LIB_OBJS += <name>
```

Object files will be used in builds of all libraries.

```
LIB_OBJS_<osclass> += <name>
```

Object files will be used in builds of all libraries for archs of the specified osclass.

```
LIB_OBJS_DEFAULT += <name>
```

Object files will be used in builds of all libraries for archs without a LIB_OBJS_<osclass> definition specified.

```
<libname>_OBJS += <name>
```

Object files will be used for all builds of the named library)

```
<libname>_OBJS_<osclass> += <name>
```

Object files will be used in builds of the library for archs with the specified osclass.

```
<libname>_OBJS_DEFAULT += <name>
```

Object files will be used in builds of the library for archs without a <libname>_OBJS_<osclass> definition specified.

Combined object files, from R3.13 built modules and applications which have file names that do not include a ".o" or ".obj" suffix (e.g. xyzLib) are defined as follows:

```
USR_OBJLIBS += <name>
```

Combined object files will be used in builds of all libraries and products.

```
USR_OBJLIBS_<osclass> += <name>
```

Combined object files will be used in builds of all libraries and products for archs of the specified osclass.

```
USR_OBJLIBS_DEFAULT += <name>
```

Combined object files will be used in builds of all libraries and products for archs without a USR_OBJLIBS_<osclass> definition specified.

```
LIB_OBJLIBS += <name>
```

Combined object files will be used in builds of all libraries.

```
LIB_OBJLIBS_<osclass> += <name>
```

Combined object files will be used in builds of all libraries for archs of the specified osclass.

```
LIB_OBJLIBS_DEFAULT += <name>
```

Combined object files will be used in builds of all libraries for archs without a LIB_OBJLIBS_<osclass> definition specified.

```
<libname>_OBJLIBS += <name>
```

Combined object files will be used for all builds of the named library.

```
<libname>_OBJLIBS_<osclass> += <name>
```

Combined object files will be used in builds of the library for archs with the specified osclass.

```
<libname>_OBJLIBS_DEFAULT += <name>
```

Combined object files will be used in builds of the library for archs without a <libname>_OBJLIBS_<osclass> definition specified.

```
<libname>_LDOBJS += <name>
```

Combined object files will be used for all builds of the named library. (deprecated)

```
<libname>_LDOBJS_<osclass> += <name>
```

Combined object files will be used in builds of the library for archs with the specified osclass. (deprecated)

```
<libname>_LDOBJS_DEFAULT += <name>
```

Combined object files will be used in builds of the library for archs without a <libname>_LDOBJS_<osclass> definition specified. (deprecated)

LIBOBJS definitions

Previous versions of epics (3.13 and before) accepted definitions like:

```
LIBOBJS += $(<support>_BIN)/xxx.o
```

These are gathered together in files such as baseLIBOBJS. To use such definitions include the lines:

```
-include ../baseLIBOBJS
<libname>_OBJS += $(LIBOBJS)
```

Note:

vxWorks applications created by `makeBaseApp.pl` from 3.14 Base releases no longer have a file named baseLIBOBJS. Base record and device support now exists in archive libraries.*

Specifying dependant libraries to be linked when creating a library

For each library name specified which is not a system library nor a library from an EPICS top defined in the `configure/RELEASE` file, a <name>_DIR definition must be present in the Makefile to specify the location of the library.

Library names, which must not have a directory and "lib" prefix nor a suffix, are defined as follows:

```
LIB_LIBS += <name>
```

Libraries to be used when linking all defined libraries.

```
LIB_LIBS_<osclass> += <name>
```

Libraries to be used or all archs of the specified osclass when linking all defined libraries.

```
LIB_LIBS_DEFAULT += <name>
```

Libraries to be used for any arch that does not have a LIB_LIBS_<osclass> definition when linking all defined libraries.

```
USR_LIBS += <name>
```

Libraries to be used when linking all defined products and libraries.

```
USR_LIBS_<osclass> += <name>
```

Libraries to be used or all archs of the specified osclasswhen linking all defined products and libraries.

```
USR_LIBS_DEFAULT += <name>
```

Libraries to be used for any arch that does not have a USR_LIBS_<osclass> definition when linking all defined products and libraries.

```
<libname>_LIBS += <name>
```

Libraries to be used for linking the named library.

```
<libname>_LIBS_<osclass> += <name>
```

Libraries will be used for all archs of the specified osclass for linking named library.

```
<libname>_LIBS_DEFAULT += <name>
```

Libraries to be used for any arch that does not have a <libname>_LIBS_<osclass> definition when linking named library.

```
<libname>_SYS_LIBS += <name>
```

System libraries to be used for linking the named library.

```
<libname>_SYS_LIBS_<osclass> += <name>
```

System libraries will be used for all archs of the specified osclass for linking named library.

```
<libname>_SYS_LIBS_DEFAULT += <name>
```

System libraries to be used for any arch that does not have a <libname>_LIBS_<osclass> definition when linking named library.

The order of dependant libraries

Dependant library names appear in the following order on a library link line:

1. <libname>_LIBS
2. <libname>_LIBS_<osclass> or <libname>_LIBS_DEFAULT
3. LIB_LIBS
4. LIB_LIBS_<osclass> or LIB_LIBS_DEFAULT
5. USR_LIBS
6. USR_LIBS_<osclass> or USR_LIBS_DEFAULT
7. <libname>_SYS_LIBS
8. <libname>_SYS_LIBS_<osclass> or <libname>_SYS_LIBS_DEFAULT
9. LIB_SYS_LIBS
10. LIB_SYS_LIBS_<osclass> or LIB_SYS_LIBS_DEFAULT
11. USR_SYS_LIBS
12. USR_SYS_LIBS_<osclass> or USR_SYS_LIBS_DEFAULT

Specifying library DLL file names (deprecated)

WIN32 libraries require all external references to be resolved, so if a library contains references to items in other DLL libraries, these DLL library names must be specified (without directory prefix and without ".dll" suffix) as follows:

```
DLL_LIBS += <name>
```

These DLLs will be used for all libraries.

```
<libname>_DLL_LIBS += <name>
```

These DLLs will be used for the named library.

Each <name> must have a corresponding <name>_DIR definition specifying its directory location.

Specifying shared library version number

A library version number can be specified when creating a shared library as follows:

```
SHRLIB_VERSION = <version>
```

On WIN32 this results in /version:$(SHRLIB_VERSION) link option. On Unix type hosts .$(SHRLIB_VERSION) is appended to the shared library name and a symbolic link is created for the unversioned library name. $(EPICS_VERSION).$(EPICS_REVISION) is the default value for SHRLIB_VERSION.

Library example:

```
LIBRARY_vxWorks += vxWorksOnly
LIBRARY_IOC += iocOnly
LIBRARY_HOST += hostOnly
LIBRARY += all
vxWorksOnly_OBJS += $(LINAC_BIN)/vxOnly1
vxWorksOnly_SRCS += vxOnly2.c
iocOnly_OBJS += $(LINAC_BIN)/iocOnly1
iocOnly_SRCS += iocOnly2.cpp
hostOnly_OBJS += $(LINAC_BIN)/host1
all_OBJS += $(LINAC_BIN)/all1
all_SRCS += all2.cpp
```

If the architectures defined in <top>/configure are solaris-sparc and vxWorks-68040 and LINAC is defined in the <top>/configure/RELEASE file, then the following libraries will be created:

- $(INSTALL_LOCATION)/bin/vxWork-68040/libvxWorksOnly.a : $(LINAC_BIN)/vxOnly1.o vxOnly2.o
- $(INSTALL_LOCATION)/bin/vxWork-68040/libiocOnly.a : $(LINAC_BIN/iocOnly1.o iocOnly2.o
- $(INSTALL_LOCATION)/lib/solaris-sparc/libiocOnly.a : $(LINAC_BIN)/iocOnly1.o iocOnly2.o
- $(INSTALL_LOCATION)/lib/solaris-sparc/libhostOnly.a : $(LINAC_BIN)/host1.o
- $(INSTALL_LOCATION)/bin/vxWork-68040/liball.a : $(LINAC_BIN)/all1.o all2.o
- $(INSTALL_LOCATION)/lib/solaris-sparc/liball.a : $(LINAC_BIN)/all1.o all2.o

Loadable libraries

Loadable libraries are regular libraries which are not required to have all symbols resolved during the build. The intent is to create dynamic plugins so no archive library is created. Source file, object files, and dependant libraries are specified in exactly the same way as for regular libraries.

Any of the following can be specified:

```
LOADABLE_LIBRARY += <name>
```

The <name> loadable library will be created for every target arch.

```
LOADABLE_LIBRARY_<osclass> += <name>
```

Loadable library <name> will be created for all archs of the specified osclass.

```
LOADABLE_LIBRARY_DEFAULT += <name>
```

Loadable library <name> will be created for any arch that does not have a LOADABLE_LIBRARY_<osclass> definition

```
LOADABLE_LIBRARY_HOST += <name>
```

Loadable library <name> will be created for HOST type archs.

```
LOADABLE_LIBRARY_HOST_<osclass> += <name>
```

Loadable library <name> will be created for all HOST type archs of the specified osclass.

```
LOADABLE_LIBRARY_HOST_DEFAULT += <name>
```

Loadable library <name> will be created for any HOST type arch that does not have a LOADABLE_LIBRARY_HOST_<osclass> definition

Combined object libraries (VxWorks only)

Combined object libraries are regular combined object files which have been created by linking together multiple object files. OBJLIB specifications in the Makefile create a combined object file and a corresponding munch file for vxWorks target architectures only. Combined object libraries have a Library.o suffix. It is possible to generate and install combined object libraries by using definitions:

```
OBJLIB += <name>
OBJLIB_vxWorks += <name>
OBJLIB_SRCS += <srcname1> <srcname2> ...
OBJLIB_OBJS += <objname1> <objname2> ...
```

These definitions result in the combined object file <name>Library.o and its corresponding <name>Library.munch munch file being built for each vxWorks architecture from source/object files in the OBJLIB_SRCS/OBJLIB_OBJS definitions. The combined object file and the munch file are installed into the $(INSTALL_LOCATION)/bin/<arch> directory.

Object Files

It is possible to generate and install object files by using definitions:

```
OBJS += <name>
OBJS_<osclass> += <name>
OBJS_DEFAULT += <name>
OBJS_IOC += <name>
OBJS_IOC_<osclass> += <name>
OBJS_IOC_DEFAULT += <name>
OBJS_HOST += <name>
OBJS_HOST_<osclass> += <name>
OBJS_HOST_DEFAULT += <name>
```

These will cause the specified file to be generated from an existing source file for the appropriate target arch and installed into `$(INSTALL_LOCATION)/bin/<arch>`.

The following Makefile will create the abc object file for all target architectures, the def object file for all target archs except vxWorks, and the xyz object file only for the vxWorks target architecture and install them into the appropriate `$(INSTALL_LOCATION)/bin/<arch>` directory.

```
TOP=../../..
include $(TOP)/configure/CONFIG
OBJS += abc
OBJS_vxWorks += xyz
OBJS_DEFAULT += def
include $(TOP)/configure/RULES
```

State Notation Programs

A state notation program file can be specified as a source file in any SRC definition. For example:

```
<prodname>_SRCS += <name>.stt
```

The state notation compiler snc will generate the file `<name>.c` from the state notation program file `<name>.stt`. This C file is compiled and the resulting object file is linked into the `<prodname>` product.

A state notation source file must have the extension .st or .stt. The .st file is passed through the C preprocessor before it is processed by snc.

If you have state notation language source files (.stt and .st files), the module seq must be built and `SNCSEQ` defined in the `RELEASE` file. If the state notation language source files require c preprocessing before conversion to c source (.st files), gcc must be in your path.

Scripts, etc.

Any of the following can be specified:

```
SCRIPTS += <name>
```

A script will be installed from the src directory to the $(INSTALL_LOCATION)/bin/<arch> directories.

```
SCRIPTS_<osclass> += <name>
```

Script <name> will be installed for all archs of the specified osclass.

```
SCRIPTS_DEFAULT += <name>
```

Script <name> will be installed for any arch that does not have a SCRIPTS_<osclass> definition

```
SCRIPTS_IOC += <name>
```

Script <name> will be installed for IOC type archs.

```
SCRIPTS_IOC_<osclass> += <name>
```

Script <name> will be installed for all IOC type archs of the specified osclass.

```
SCRIPTS_IOC_DEFAULT += <name>
```

Script <name> will be installed for any IOC type arch that does not have a SCRIPTS_IOC_<osclass> definition

```
SCRIPTS_HOST += <name>
```

Script <name> will be installed for HOST type archs.

```
SCRIPTS_HOST_<osclass> += <name>
```

Script <name> will be installed for all HOST type archs of the specified osclass.

```
SCRIPTS_HOST_DEFAULT += <name>
```

Script <name> will be installed for any HOST type arch that does not have a SCRIPTS_HOST_<osclass> definition

Definitions of the form:

```
SCRIPTS_<osclass> += <name1>
SCRIPTS_DEFAULT += <name2>
```

results in the script being installed from the src directory to the $(INSTALL_LOCATION)/bin/<arch> directories for all target archs of the specified os class and the script installed into the $(INSTALL_LOCATION)/bin/<arch> directories of all other target archs.

Include files

A definition of the form:

```
INC += <name>.h
```

results in file <name>.h being installed or created and installed to the $(INSTALL_LOCATION)/include directory.

Definitions of the form:

```
INC_DEFAULT += <name>.h
INC_<osclass> += <name>.h
```

results in file <name>.h being installed or created and installed into the appropriate $(INSTALL_LOCATION)/include/os/<osclass> directory.

Html and Doc files

A definition of the form:

```
HTMLS_DIR = <dirname>
HTMLS += <name>
```

results in file <name> being installed from the src directory to the $(INSTALL_LOCATION)/html/<dirname> directory.

A definition of the form:

```
DOCS += <name>
```

results in file <name> being installed from the src directory to the $(INSTALL_LOCATION)/doc directory.

Templates

Adding definitions of the form

```
TEMPLATES_DIR = <dirname>
TEMPLATES += <name>
```

results in the file <name> being installed from the src directory to the $(INSTALL_LOCATION)/templates/<dirname> directory. If a directory structure of template files is to be installed, the template file names may include a directory prefix.

Lex and yacc

If a <name>.c source file specified in a Makefile definition is not found in the source directory, gnumake will try to build it from <name>.y and <name>_lex.l files in the source directory. Lex converts a <name>.l Lex code file to a lex.yy.c file which the build rules renames to <name>.c. Yacc converts a <name>.y yacc code file to a y.tab.c file, which the build rules renames to <name>.c. Optionally yacc can create a y.tab.h file which the build rules renames to <name>.h.

Products

A product executable is created for each and installed into $(INSTALL_LOCATION)/bin/<arch> by specifying its name and the name of either the object or source files containing code for the product. An object or source file name can appear with or without a directory prefix. Object files should contain a directory prefix. If the file has a directory prefix e.g. $(EPICS_BASE_BIN), the file is taken from the specified location. If a directory prefix is not present, make will look in the source directories for a file with the specified name or try build it using existing rules. An executable filename suffix appropriate for the target arch (e.g. .exe) may be appended to the filename when the file is created.

PROD specifications in the Makefile for vxWorks target architectures create a combined object file with library references resolved and a corresponding .munch file.

```
PROD_HOST += <name>
<name>_SRC += <srcname>.c
```

results in the executable <name> being built for each HOST architecture, <arch>, from a <srcname>.c file. Then <name> is installed into the $(INSTALL_LOCATION)/bin/<arch> directory.

Specifying the product name.

Any of the following can be specified:

```
PROD += <name>
```

Product <name> will be created for every target arch.

```
PROD_<osclass> += <name>
```

Product <name> will be created for all archs of the specified osclass.

```
PROD_DEFAULT += <name>
```

Product <name> will be created for any arch that does not have a PROD_<osclass> definition

```
PROD_IOC += <name>
```

Product <name> will be created for IOC type archs.

```
PROD_IOC_<osclass> += <name>
```

Product <name> will be created for all IOC type archs of the specified osclass.

```
PROD_IOC_DEFAULT += <name>
```

Product <name> will be created for any IOC type arch that does not have a PROD_IOC_<osclass> definition

```
PROD_HOST += <name>
```

Product <name> will be created for HOST type archs.

```
PROD_HOST_<osclass> += <name>
```

Product <name> will be created for all HOST type archs of the specified osclass.

```
PROD_HOST_DEFAULT += <name>
```

Product <name> will be created for any HOST type arch that does not have a PROD_HOST_<osclass> definition

Specifying product object file names

Object files which have filenames with a ".o" or ".obj" suffix are defined as follows and can be specified without the suffix but should have the directory prefix

```
USR_OBJS += <name>
```

Object files will be used in builds of all products and libraries

```
USR_OBJS_<osclass> += <name>
```

Object files will be used in builds of all products and libraries for archs with the specified osclass.

```
USR_OBJS_DEFAULT += <name>
```

Object files will be used in builds of all products and libraries for archs without a USR_OBJS_<osclass> definition specified.

```
PROD_OBJS += <name>
```

Object files will be used in builds of all products

```
PROD_OBJS_<osclass> += <name>
```

Object files will be used in builds of all products for archs with the specified osclass.

```
PROD_OBJS_DEFAULT += <name>
```

Object files will be used in builds of all products for archs without a PROD_OBJS_<osclass> definition specified.

```
<prodname>_OBJS += <name>
```

Object files will be used for all builds of the named product

```
<prodname>_OBJS_<osclass> += <name>
```

Object files will be used in builds of the named product for archs with the specified osclass.

```
<prodname>_OBJS_DEFAULT += <name>
```

Object files will be used in builds of the named product for archs without a <prodname>_OBJS_<osclass> definition specified.

Combined object files, from R3.13 built modules and applications which have file names that do not include a ".o" or ".obj" suffix (e.g. xyzLib) are defined as follows:

```
USR_OBJLIBS += <name>
```

Combined object files will be used in builds of all libraries and products.

```
USR_OBJLIBS_<osclass> += <name>
```

Combined object files will be used in builds of all libraries and products for archs of the specified osclass.

```
USR_OBJLIBS_DEFAULT += <name>
```

Combined object files will be used in builds of all libraries and products for archs without a USR_OBJLIBS_<osclass> definition specified.

```
PROD_OBJLIBS += <name>
```

Combined object files will be used in builds of all products.

```
PROD_OBJLIBS_<osclass> += <name>
```

Combined object files will be used in builds of all products for archs of the specified osclass.

```
PROD_OBJLIBS_DEFAULT += <name>
```

Combined object files will be used in builds of all products for archs without a PROD_OBJLIBS_<osclass> definition specified.

```
<prodname>_OBJLIBS += <name>
```

Combined object files will be used for all builds of the named product.

```
<prodname>_OBJLIBS_<osclass> += <name>
```

Combined object files will be used in builds of the named product for archs with the specified osclass.

```
<prodname>_OBJLIBS_DEFAULT += <name>
```

Combined object files will be used in builds of the named product for archs without a <prodname>_OBJLIBS_<osclass> definition specified.

```
<prodname>_LDOBJS += <name>
```

Object files will be used for all builds of the named product. (deprecated)

```
<prodname>_LDOBJS_<osclass> += <name>
```

Object files will be used in builds of the name product for archs with the specified osclass. (deprecated)

```
<prodname>_LDOBJS_DEFAULT += <name>
```

Object files will be used in builds of the product for archs without a <prodname>_LDOBJS_<osclass> definition specified. (deprecated)

Specifying product source file names

Source file names, which must have a suffix, are defined as follows:

```
SRCS += <name>
```

Source files will be used for all defined libraries and products.

```
SRCS_<osclass> += <name>
```

Source files will be used for all defined libraries and products for all archs of the specified osclass.

```
SRCS_DEFAULT += <name>
```

Source files will be used for all defined libraries and products for any arch that does not have a SRCS_<osclass> definition

```
USR_SRCS += <name>
```

Source files will be used for all products and libraries.

```
USR_SRCS_<osclass> += <name>
```

Source files will be used for all defined products and libraries for all archs of the specified osclass.

```
USR_SRCS_DEFAULT += <name>
```

Source files will be used for all defined products and libraries for any arch that does not have a USR_SRCS_<osclass> definition

```
PROD_SRCS += <name>
```

Source files will be used for all products.

```
PROD_SRCS_<osclass> += <name>
```

Source files will be used for all defined products for all archs of the specified osclass.

```
PROD_SRCS_DEFAULT += <name>
```

Source files will be used for all defined products for any arch that does not have a PROD_SRCS_<osclass> definition

```
<prodname>_SRCS += <name>
```

Source file will be used for the named product.

```
<prodname>_SRCS_<osclass> += <name>
```

Source files will be used for named product for all archs of the specified osclass.

```
<prodname>_SRCS_DEFAULT += <name>
```

Source files will be used for named product for any arch that does not have a <prodname>_SRCS_<osclass> definition

Specifying libraries to be linked when creating the product

For each library name specified which is not a system library nor a library from EPICS base, a <name>_DIR definition must be present in the Makefile to specify the location of the library.

Library names, which must not have a directory and "lib" prefix nor a suffix, are defined as follows:

```
PROD_LIBS += <name>
```

Libraries to be used when linking all defined products.

```
PROD_LIBS_<osclass> += <name>
```

Libraries to be used or all archs of the specified osclass when linking all defined products.

```
PROD_LIBS_DEFAULT += <name>
```

Libraries to be used for any arch that does not have a PROD_LIBS_<osclass> definition when linking all defined products.

```
USR_LIBS += <name>
```

Libraries to be used when linking all defined products.

```
USR_LIBS_<osclass> += <name>
```

Libraries to be used or all archs of the specified osclasswhen linking all defined products.

```
USR_LIBS_DEFAULT += <name>
```

Libraries to be used for any arch that does not have a USR_LIBS_<osclass> definition when linking all defined products.

```
<prodname>_LIBS += <name>
```

Libraries to be used for linking the named product.

```
<prodname>_LIBS_<osclass> += <name>
```

Libraries will be used for all archs of the specified osclass for linking named product.

```
<prodname>_LIBS_DEFAULT += <name>
```

Libraries to be used for any arch that does not have a <prodname>_LIBS_<osclass> definition when linking named product.

```
SYS_PROD_LIBS += <name>
```

System libraries to be used when linking all defined products.

```
SYS_PROD_LIBS_<osclass> += <name>
```

System libraries to be used for all archs of the specified osclass when linking all defined products.

```
SYS_PROD_LIBS_DEFAULT += <name>
```

System libraries to be used for any arch that does not have a PROD_LIBS_<osclass> definition when linking all defined products.

```
<prodname>_SYS_LIBS += <name>
```

System libraries to be used for linking the named product.

```
<prodname>_SYS_LIBS_<osclass> += <name>
```

System libraries will be used for all archs of the specified osclass for linking named product.

```
<prodname>_SYS_LIBS_DEFAULT += <name>
```

System libraries to be used for any arch that does not have a <prodname>_LIBS_<osclass> definition when linking named product.

The order of dependant libraries

Dependant library names appear in the following order on a product link line:

1. <prodname>_LIBS
2. <prodname>_LIBS_<osclass> or <prodname>_LIBS_DEFAULT
3. PROD_LIBS
4. PROD_LIBS_<osclass> or PROD_LIBS_DEFAULT
5. USR_LIBS
6. USR_LIBS_<osclass> or USR_LIBS_DEFAULT
7. <prodname>_SYS_LIBS
8. <prodname>_SYS_LIBS_<osclass> or <prodname>_SYS_LIBS_DEFAULT
9. PROD_SYS_LIBS
10. PROD_SYS_LIBS_<osclass> or PROD_SYS_LIBS_DEFAULT
11. USR_SYS_LIBS
12. USR_SYS_LIBS_<osclass> or USR_SYS_LIBS_DEFAULT

Specifying product version number

On WIN32 only a product version number can be specified as follows:

```
PROD_VERSION += <version>
```

This results in /version:$(PROD_VERSION) link option.

Generate version header

A header can be generated which defines a single string macro with an automatically generated identifier. The default is the ISO 8601 formatted time of the build. A revision id is used if a supported version control system is present. This will typically be used to make an automatically updated source version number visible at runtime (eg. with a stringin record).

To enable this the variable `GENVERSION` must be set with the desired name of the generated header. By default this variable is empty and no header will be generated. If specified, this variable must be set before `configure/RULES` is included.

It is also necessary to add an explicit dependency for each source file which includes the generated header.

An Makefile which generates a version header named `myversion.h` included by `devVersionString.c` would have the following.

```
TOP=../..
include $(TOP)/configure/CONFIG
# ... define PROD or LIBRARY names sometarget
sometarget_SRCS = devVersionString.c
GENVERSION = myversion.h
include $(TOP)/configure/RULES
# for each source file
devVersionString$(DEP): $(GENVERSION)
```

The optional variables `GENVERSIONMACRO` and `GENVERSIONDEFAULT` give the name of the C macro which will be defined in the generated header, and its default value if no version control system is being used. To avoid conflicts, the macro name must be changed from its default `MODULEVERSION` if the version header is to be installed.

Product static builds

Product executables can be linked with either archive versions or shared versions of EPICS libraries. Shared versions of system libraries will always be used in product linking. The definition of `STATIC_BUILD` (`YES`/`NO`) in `base/configure/CONFIG_SITE` determines which EPICS libraries to use. When `STATIC_BUILD` is `NO`, shared libraries will be used. (`SHARED_LIBRARIES` must be set to `YES`.) The default definition for `STATIC_BUILD` in the EPICS base `CONFIG_SITE` distribution file is `NO`. A `STATIC_BUILD` definition in a Makefile will override the definition in `CONFIG_SITE`. Static builds may not be possible on all systems. For static builds, all nonsystem libraries must have an archive version, and this may not be true form all libraries.

Test Products

Test products are product executables that are created but not installed into `$(INSTALL_LOCATION)/bin/<arch>` directories. Test product libraries, source, and object files are specified in exactly the same way as regular products.

Any of the following can be specified:

```
TESTPROD += <name>
```

Test product <name> will be created for every target arch.

```
TESTPROD_<osclass> += <name>
```

Test product <name> will be created for all archs of the specified osclass.

```
TESTPROD_DEFAULT += <name>
```

Test product <name> will be created for any arch that does not have a TESTPROD_<osclass> definition

```
TESTPROD_IOC += <name>
```

Test product <name> will be created for IOC type archs.

```
TESTPROD_IOC_<osclass> += <name>
```

Test product <name> will be created for all IOC type archs of the specified osclass.

```
TESTPROD_IOC_DEFAULT += <name>
```

Test product <name> will be created for any IOC type arch that does not have a TESTPROD_IOC_<osclass> definition

```
TESTPROD_HOST += <name>
```

Test product <name> will be created for HOST type archs.

```
TESTPROD_HOST_<osclass> += <name>
```

Test product <name> will be created for all HOST type archs of the specified osclass.

```
TESTPROD_HOST_DEFAULT += <name>
```

Test product <name> will be created for any HOST type arch that does not have a TESTPROD_HOST_<osclass> definition

Test Scripts

Test scripts are perl scripts whose names end in .t that get executed to satisfy the runtests make target. They are run by the perl Test::Harness library, and should send output to stdout following the Test Anything Protocol. Any of the following can be specified, although only TESTSCRIPTS_HOST is currently useful:

```
TESTSCRIPTS += <name>
```

Test script <name> will be created for every target arch.

```
TESTSCRIPTS_<osclass> += <name>
```

Test script <name> will be created for all archs of the specified osclass.

```
TESTSCRIPTS_DEFAULT += <name>
```

Test script <name> will be created for any arch that does not have a TESTSCRIPTS_<osclass> definition

```
TESTSCRIPTS_IOC += <name>
```

Test script <name> will be created for IOC type archs.

```
TESTSCRIPTS_IOC_<osclass> += <name>
```

Test script <name> will be created for all IOC type archs of the specified osclass.

```
TESTSCRIPTS_IOC_DEFAULT += <name>
```

Test script <name> will be created for any IOC type arch that does not have a TESTSCRIPTS_IOC_<osclass> definition

```
TESTSCRIPTS_HOST += <name>
```

Test script <name> will be created for HOST type archs.

```
TESTSCRIPTS_HOST_<osclass> += <name>
```

Test script <name> will be created for all HOST type archs of the specified osclass.

```
TESTSCRIPTS_HOST_DEFAULT += <name>
```

Test script <name> will be created for any HOST type arch that does not have a TESTSCRIPTS_HOST_<osclass> definition.

If a name in one of the above variables matches a regular executable program name (normally generated as a test product) with .t appended, a suitable perl script will be generated that will execute that program directly; this makes it simple to run programs that use the epicsUnitTest routines in libCom. A test script written in Perl with a name ending .plt will be copied into the O.<arch> directory with the ending changed to .t; such scripts will usually use the perl Test::Simple or Test::More libraries.

Miscellaneous Targets

A definition of the form:

```
TARGETS += <name>
```

results in the file <name> being built in the O.<arch> directory from existing rules and files in the source directory. These target files are not installed.

Installing Other Binaries

Definitions of the form:

```
BIN_INSTALLS += <name>
BIN_INSTALLS += <dir>/<name>
BIN_INSTALLS_DEFAULT += <name>
BIN_INSTALLS_<osclass> += <name>
```

will result in the named files being installed to the appropriate $(INSTALL_LOCATION)/bin/<arch> directory. The file <name> can appear with or without a directory prefix. If the file has a directory prefix e.g. $(EPICS_BASE_BIN), it is copied from the specified location. If a directory prefix is not present, make will look in the source directory for the file.

Installing Other Libraries

Definitions of the form:

```
LIB_INSTALLS += <name>
LIB_INSTALLS += <dir>/<name>
LIB_INSTALLS_DEFAULT += <name>
LIB_INSTALLS_<osclass> += <name>
```

result in files being installed to the appropriate `$(INSTALL_LOCATION)/lib/<arch>` directory. The file `<name>` can appear with or without a directory prefix. If the file has a directory prefix e.g. `$(EPICS_BASE_LIB)`, it is copied from the specified location. If a directory prefix is not present, make will look in the source directory for the file.

Win32 resource files

Definitions of the following forms result in resource files (`*.res` files) being created from the specified `*.rc` resource definition script files and linked into the prods and/or libraries.

```
RCS += <name>
RCS_<osclass> += <name>
```

Resource definition script files for all products and libraries.

```
PROD_RCS += <name>
PROD_RCS_<osclass> += <name>
PROD_RCS_DEFAULT += <name>
```

Resource definition script files for all products.

```
LIB_RCS += <name>
LIB_RCS_<osclass> += <name>
LIB_RCS_DEFAULT += <name>
```

Resource definition script files for all libraries.

```
<name>_RCS += <name>
<name>_RCS_<osclass> += <name>
<name>_RCS_DEFAULT += <name>
```

Resource definition script files for specified product or library.

TCL libraries

Definitions of the form:

```
TCLLIBNAME += <name>
TCLINDEX += <name>
```

result in the specified tcl files being installed to the `$(INSTALL_LOCATION)/lib/<arch>` directory.

Java class files

Java class files can be created by the `javac` tool into `$(INSTALL_JAVA)` or into the `O.Common` subdirectory, by specifying the name of the java class file in the Makefile. Command line options for the `javac` tool can be specified. The configuration files set the java c option `-sourcepath .:..:../...`

Any of the following can be specified:

```
JAVA += <name>.java
```

The `<name>.java` file will be used to create the `<name>.class` file in the `$(INSTALL_JAVA)` directory.

```
TESTJAVA += <name>.java
```

The `<name>.java` files will be used to create the `<name>.class` file in the `O.Common` subdirectory.

```
USR_JAVACFLAGS += <name>
```

The `javac` option `<name>` will be used on the `javac` command lines.

Example 1

In this example, three class files are created in `$(INSTALL_LOCATION)/javalib/mytest`. The `javac` depreciation flag is used to list the description of each use or override of a deprecated member or class.

```
JAVA = mytest/one.java
JAVA = mytest/two.java
JAVA = mytest/three.java
USR_JAVACFLAGS = -deprecation
```

Example 2

In this example, the `test.class` file is created in the `O.Common` subdirectory.

```
TESTJAVA = test.java
```

Java jar file

A single java jar file can be created using the java jar tool and installed into `$(INSTALL_JAVA)` (i.e. `$(INSTALL_LOCATION)/javalib`) by specifying its name, and the names of its input files to be included in the created jar file. The jar input file names must appear with a directory prefix.

Any of the following can be specified:

```
JAR += <name>
```

The `<name>` jar file will be created and installed into the `$(INSTALL_JAVA)` directory.

```
JAR_INPUT += <name>
```

Names of images, audio files and classes files to be included in the jar file.

```
JAR_MANIFEST += <name>
```

The preexisting manifest file will be used for the created jar file.

```
JAR_PACKAGES += <name>
```

Names of java packages to be installed and added to the created jar file.

Example 3

In this example, all the class files created by the current Makefile's `JAVA +=` definitions, are placed into a file named `mytest1.jar`. A manifest file will be automatically generated for the jar.

Note

`$(INSTALL_CLASSES)` is set to `$(addprefix $(INSTALL_JAVA)/,$(CLASSES))` in the EPICS base configure files.

```
JAR = mytest1.jar
JAR_INPUT = $(INSTALL_CLASSES)
```

Example 4

In this example, three class files are created and placed into a new jar archive file named `mytest2.jar`. An existing manifest file, `mytest2.mf` is put into the new jar file.

```
JAR = mytest2.jar
JAR_INPUT = $(INSTALL_JAVA)/mytest/one.class
JAR_INPUT = $(INSTALL_JAVA)/mytest/two.class
JAR_INPUT = $(INSTALL_JAVA)/mytest/three.class
JAR_MANIFEST = mytest2.mf
```

Java native method C header files

A C header files for use with java native methods will be created by the `javah` tool in the `O.Common` subdirectory by specifying the name of the header file to be created. The name of the java class file used to generate the header is derived from the name of the header file. Underscores _ are used as a header file name delimiter. Command line options for the `javah` tool can be specified.

Any of the following can be specified:

```
JAVAINC += <name>.h
```

The <name>.h header file will be created in the `O.Common` subdirectory.

```
USR_JAVAHFLAGS += <name>
```

The `javah` option <name> will be used on the `javah` tool command line.

Example 5

In this example, the C header xx_yy_zz.h will be created in the $(COMMON_DIR) subdirectory from the class xx.yy.zz (i.e. the java class file $(INSTALL_JAVA)/xx/yy/zz.class)). The option -old will tell javah to create old JDK1.0 style header files.

```
JAVAINC = xx_yy_zz.h
USR_JAVAHFLAGS = -old
```

User Created CONFIG* and RULES* files

Module developers can now create new CONFIG and RULES* files ia a <top> application source directory. These new CONFIG* or RULES* files will be installed into the directory $(INSTALL_LOCATION)/cfg by including lines like the following Makefile line:

```
CFG += CONFIG_MY1 RULES_MY1
```

The build will install the new files CONFIG_MY1 and RULES_MY1 into the $(INSTALL_LOCATION)/cfg directory.

Files in a $(INSTALL_LOCATION)/cfg directory are now included during a build by so that the definitions and rules in them are available for use by later src directory Makefiles in the same module or by other modules with a RELEASE line pointing to the TOP of this module.

User Created File Types

Module developers can now define a new type of file, e.g. ABC, so that files of type ABC will be installed into a directory defined by INSTALL_ABC. This is done by creating a new CONFIG_<name> file, e.g. CONFIG_ABC, with the following lines:

```
FILE_TYPE += ABC
INSTALL_ABC = $(INSTALL_LOCATION)/abc
```

The INSTALL_ABC directory should be a subdirectory of $(INSTALL_LOCATION). The file type ABC should be target architecture independent (alh files, medm files, edm files.

Optional rules necessary for files of type ABC should be put in a RULES_ABC file.

The module developer installs new CONFIG_ABC and RULES_ABC files for the new file type into the directory $(INSTALL_LOCATION)/cfg by including the following Makefile line:

```
CFG += CONFIG_ABC RULES_ABC
```

Files of type ABC are installed into INSTALL_ABC directory by adding a line like the following to a Makefile.

```
ABC += <filename1> <filename2> <filename3>
```

Since the files in $(INSTALL_LOCATION)/cfg directory are now included by the base config files, the ABC += definition lines are available for use by later src directory Makefiles in the same module or by other modules with a RELEASE line pointing to the TOP of this module.

Assemblies

A single output file is generated from assembling specified snippet files. Snippet file names start with numbers and are sorted when the snippets are concatenated: first by the number, then alphabetical by the remaining part of the name. (This mechanism is conceptually similar to the Linux convention of collecting configuration file snippets in *.d directories.)

Snippets with file names not starting with a number or ending in '~' are ignored. The specified snippets are processed in the order they appear on the command line. Multiple snippets with the same number are concatenated. "Commands" (tags in the snippet name) can be used to control the treatment of snippets with the same number:

- D - Default. Snippet is treated as a default, which is replaced (overwritten) by any other snippet with the same number.
- R - Replace. Snippet is replacing (overwriting) already processed snippets with the same number.

Specification of the target file is different for architecture dependent or independent files.

```
COMMON_ASSEMBLIES += st.cmd
ASSEMBLIES += mytool.rc
```

Snippet files are configured specifically (relative or absolute path) or as patterns (searched relative to all source directories).

```
mytool.rc_SNIPPETS += ../rc.d/10_head ../rc.d/20_init
st.cmd_PATTERN += st.cmd.d/*
```

Macros

The following macros can be used in snippets, and will be replaced by the current value when assembling is done.

- `DATETIME` Date and time of the build
- `USERNAME` Name of the user running the build
- `HOST` Name of the host on which the build is run
- `OUTPUTFILE` Name of the generated file
- `SNIPPETFILE` Name of the current snippet

Example 6

This mechanism can be used to create an IOC startup file from snippets in a global and an application specific directory, allowing applications to add commands to different phases of the IOC startup by dropping appropriately numbered snippets into the directory.

Given the following directories and snippets:

```
/global/st.cmd.d/
├── D10_init
├── 20_environment
├── 30_drivers
├── D40_settings
└── 70_start-ioc
```

```
../st.cmd.d/
├── D10_init
├── 40_settings
├── 40_settings~       # backup file
├── 30_another-driver
└── R70_start-my-ioc
```

And the following Makefile declaration:

```
SCRIPTS += $(COMMON_DIR)/st.cmd
COMMON_ASSEMBLIES += st.cmd
st.cmd_SNIPPETS += $(wildcard /global/st.cmd.d/*)
st.cmd_PATTERN += st.cmd.d/*
```

The build will create and install a st.cmd script using the following snippets:

```
+---------+---------------------------+-----------------------------+
| Source  | Snippet                   | Comment                     |
+=========+===========================+=============================+
| L G     | 10_init 20_environment    | L default resets the G      |
|         |                           | default                     |
+---------+---------------------------+-----------------------------+
| L G     | 30_another-driver         | implicit addition,          |
|         | 30_drivers                | alphabetical sorting        |
+---------+---------------------------+-----------------------------+
| L       | 40_settings               | replacing a default,        |
|         |                           | ignoring backup file        |
+---------+---------------------------+-----------------------------+
| #####   | 70_start-my-ioc           | explicit replace            |
|  L {#l} |                           |                             |
+---------+---------------------------+-----------------------------+
```

Table of Makefile definitions

Definitions given below containing <osclass> are used when building for target archs of a specific osclass, and the <osclass> part of the name should be replaced by the desired osclass, e.g. solaris, vxWorks, etc. If a _DEFAULT setting is given but a particular <osclass> requires that the default not apply and there are no items in the definition that apply for that <osclass>, the value -nil- should be specified in the relevant Makefile definition.

```
+---------------------------------+---------------------------------+
| Build Option                    | Description                     |
+=================================+=================================+
| Products to be built (host type |                                 |
| archs only)                     |                                 |
+---------------------------------+---------------------------------+
| PROD                            | products (names without         |
|                                 | execution suffix) to build and  |
|                                 | install. Specify xyz to build   |
|                                 | executable xyz on Unix and      |
|                                 | xyz.exe on WIN32                |
+---------------------------------+---------------------------------+
| PROD_<osclass>                  | os class specific products to   |
```

(continues on next page)

(continued from previous page)

	build and install for <osclass> archs only
PROD_DEFAULT	products to build and install for archs with no PROD_<osclass> specified
PROD_IOC	products to build and install for ioc type archs
PROD_IOC_<osclass>	os specific products to build and install for ioc type archs
PROD_IOC_DEFAULT	products to build and install for ioc type arch systems with no PROD_IOC_<osclass> specified
PROD_HOST	products to build and install for host type archs.
PROD_HOST_<osclass>	os class specific products to build and install for <osclass> type archs
PROD_HOST_DEFAULT Test products to be built	products to build and install for arch with no PROD_HOST_<osclass> specified
TESTPROD	test products (names without execution suffix) to build but not install
TESTPROD_<osclass>	os class specific test products to build but not install
TESTPROD_DEFAULT	test products to build but not install for archs with no TESTPROD_<osclass> specified
TESTPROD_IOC	test products to build and install for ioc type archs
TESTPROD_IOC_<osclass>	os specific test products to build and install for ioc type archs
TESTPROD_IOC_DEFAULT	test products to build and install for ioc type arch systems with no

(continues on next page)

(continued from previous page)

	TESTPROD_IOC_<osclass> specified
TESTPROD_HOST	testproducts to build and install for host type archs.
TESTPROD_HOST_<osclass>	os class specific testproducts to build and install for <osclass> type archs
TESTPROD_HOST_DEFAULT Test scripts to be built	test products to build and install for arch with no TESTPROD_HOST_<osclass> specified
TESTSCRIPTS	test scripts (names with .t suffix) to build but not install
TESTSCRIPTS_<osclass>	os class specific test scripts to build but not install
TESTSCRIPTS_DEFAULT	test scripts to build but not install for archs with no TESTSCRIPTS_<osclass> specified
TESTSCRIPTS_IOC	test scripts to build and install for ioc type archs
TESTSCRIPTS_IOC_<osclass>	os specific test scripts to build and install for ioc type archs
TESTSCRIPTS_IOC_DEFAULT	test scripts to build and install for ioc type arch systems with no TESTSCRIPTS_IOC_<osclass> specified
TESTSCRIPTS_HOST	test scripts to build and install for host type archs.
TESTSCRIPTS_HOST_<osclass>	os class specific testscripts to build and install for <osclass> type archs
TESTSCRIPTS_HOST_DEFAULT Libraries to be built	test scripts to build and install for arch with no TESTSCRIPTS_HOST_<osclass> specified
LIBRARY	name of library to build and

(continues on next page)

(continued from previous page)

	install. The name should NOT include a prefix or extension e.g. specify Ca to build libCa.a on Unix, Ca.lib or Ca.dll on WIN32
LIBRARY_<osclass>	os specific libraries to build and install
LIBRARY_DEFAULT	libraries to build and install for archs with no LIBRARY_<osclass> specified
LIBRARY_IOC	name of library to build and install for ioc type archs. The name should NOT include a prefix or extension e.g. specify Ca to build libCa.a on Unix, Ca.lib or Ca.dll on WIN32
LIBRARY_IOC_<osclass>	os specific libraries to build and install for ioc type archs
LIBRARY_IOC_DEFAULT	libraries to build and install for ioc type arch systems with no LIBRARY_IOC_<osclass> specified
LIBRARY_HOST	name of library to build and install for host type archs. The name should NOT include a prefix or extension, e.g. specify Ca to build libCa.a on Unix, Ca.lib or Ca.dll on WIN32
LIBRARY_HOST_<osclass>	os class specific libraries to build and install for host type archs
LIBRARY_HOST_DEFAULT	libraries to build and install for host type arch systems with no LIBRARY_HOST_<osclass> specified
SHARED_LIBRARIES	build shared libraries? Must be YES or NO
SHRLIB_VERSION Loadable libraries to be built	shared library version number
LOADABLE_LIBRARY	name of loadable library to build and install. The name

(continues on next page)

(continued from previous page)

	should NOT include a prefix or extension e.g. specify Ca to build libCa.so on Unix and Ca.dll on WIN32
LOADABLE_LIBRARY_<osclass>	os specific loadable libraries to build and install
LOADABLE_LIBRARY_DEFAULT	loadable libraries to build and install for archs with no LOADABLE_LIBRARY_<osclass> specified
LOADABLE_LIBRARY_HOST	name of loadable library to build and install for host type archs. The name should NOT include a prefix or extension, e.g. specify test to build libtest.so on Unix and test.dll on WIN32
LOADABLE_LIBRARY_HOST_<osclass>	os class specific loadable libraries to build and install for host type archs
LOADABLE_LIBRARY_HOST_DEFAULT Combined object files (vxWorks only)	loadable libraries to build and install for host type arch systems with no LOADABLE_LIBRARY_HOST_<osclass> specified
OBJLIB	name of a combined object file library and corresponding munch file to build and install. The name will have a Library suffix appended
OBJLIB_vxWorks	same as OBJLIB
OBJLIB_SRCS	source files to build the OBJLIB
OBJLIB_OBJS Product and library source files	object files to include in OBJLIB
SRCS	source files to build all PRODs and LIBRARYs
SRCS_<osclass>	osclass specific source files to build all PRODs and LIBRARYs
SRCS_DEFAULT	source file to build all PRODs and LIBRARYs for archs with no

(continues on next page)

(continued from previous page)

	SRCS_<osclass> specified
USR_SRCS	source files to build all PRODs and LIBRARYs
USR_SRCS_<osclass>	osclass specific source files to build all PRODs and LIBRARYs
USR_SRCS_DEFAULT	source file to build all PRODs and LIBRARYs for archs with no SRCS_<osclass> specified
PROD_SRCS	source files to build all PRODs
PROD_SRCS_<osclass>	osclass specific source files to build all PRODs
PROD_SRCS_DEFAULT	source files needed to build PRODs for archs with no SRCS_<osclass> specified
LIB_SRCS	source files for building LIBRARY (e.g. LIB_SRCS=la.c lb.c lc.c)
LIB_SRCS_<osclass>	os-specific library source files
LIB_SRCS_DEFAULT	library source files for archs with no LIB_SRCS_<osclass> specified
LIBSRCS	source files for building LIBRARY (deprecated)
LIBSRCS_<osclass>	os-specific library source files (deprecated)
LIBSRCS_DEFAULT	library source files for archs with no LIBSRCS_<osclass> specified (deprecated)
<name>_SRCS	source files to build a specific PROD or LIBRARY
<name>_SRCS_<osclass>	os specific source files to build a specific PROD or LIBRARY
<name>_SRCS_DEFAULT Product and library object files	source files needed to build a specific PROD or LIBRARY for archs with no <prod>_SRCS_<osclass> specified

(continues on next page)

(continued from previous page)

USR_OBJS	object files, specified without suffix, to build all PRODs and LIBRARYs
USR_OBJS_<osclass>	osclass specific object files, specified without suffix, to build all PRODs and LIBRARYs
USR_OBJS_DEFAULT	object files, specified without suffix, needed to build PRODs and LIBRARYs for archs with no OBJS_<osclass> specified
PROD_OBJS	object files, specified without suffix, to build all PRODs
PROD_OBJS_<osclass>	osclass specific object files, specified without suffix, to build all PRODs
PROD_OBJS_DEFAULT	object files, specified without suffix, needed to build PRODs for archs with no OBJS_<osclass> specified
LIB_OBJS	object files, specified without suffix, for building all LIBRARYs (e.g. LIB_OBJS+=$(AB_BIN)/la $(AB_BIN)/lb)
LIB_OBJS_<osclass>	os-specific library object files, specify without suffix,
LIB_OBJS_DEFAULT	library object files, specified without suffix, for archs with no LIB_OBJS_<osclass> specified
<name>_OBJS	object files, specified without suffix, to build a specific PROD or LIBRARY
<name>_OBJS_<osclass>	os specific object files, specified without suffix, to build a specific PROD or LIBRARY
<name>_OBJS_DEFAULT Product and library R3.13 combined object files	object files, without suffix, needed to build a specific PROD or LIBRARY for archs with no

(continues on next page)

(continued from previous page)

	<prod>_OBJS_<osclass> specified
USR_OBJLIBS	combined object files with filenames that do not have a suffix, needed for building all PRODs and LIBRARYs (e.g. USR_OBJLIBS+=$(XYZ_BIN)/xyzLib)
USR_OBJLIBS_<osclass>	os-specific combined object files with filenames that do not have a suffix for building all PRODs and LIBRARYs
USR_OBJLIBS_DEFAULT	combined object files with filenames that do not have a suffix, for archs with no USR_OBJLIBS_<osclass> specified for building all PRODs and LIBRARYs
PROD_OBJLIBS	combined object files with filenames that do not have a suffix, needed for building all PRODs (e.g. P ROD_OBJLIBS+=$(XYZ_BIN)/xyzLib)
PROD_OBJLIBS_<osclass>	os-specific combined object files with filenames that do not have a suffix for building all PRODs
PROD_OBJLIBS_DEFAULT	combined object files with filenames that do not have a suffix, for archs with no PROD_OBJLIBS_<osclass> specified for building all PRODs
LIB_OBJLIBS	combined object files with filenames that do not have a suffix, needed for building all LIBRARYs (e.g. LIB_OBJLIBS+=$(XYZ_BIN)/xyzLib)
LIB_OBJLIBS_<osclass>	os-specific combined object files with filenames that do not have a suffix for building all LIBRARYs
LIB_OBJLIBS_DEFAULT	combined object files with filenames that do not have a

(continues on next page)

(continued from previous page)

		suffix, for archs with no LIB_OBJLIBS_<osclass> specified for building all LIBRARYs
<name>_OBJLIBS		combined object files with filenames that do not have a suffix, needed to build a specific PROD or LIBRARY
<name>_OBJLIBS_<osclass>	os specific	combined object files with filenames that do not have a suffix, to build a specific PROD or LIBRARY
<name>_OBJLIBS_DEFAULT		combined object files with filenames that do not have a suffix, needed to build a specific PROD or LIBRARY for archs with no <name>_OBJLIBS_<osclass> specified
<name>_LDOBJS		combined object files with filenames that do not have a suffix, needed to build a specific PROD or LIBRARY (deprecated)
<name>_LDOBJS_<osclass>		os specific combined object files with filenames that do not have a suffix, to build a specific PROD or LIBRARY (deprecated)
<name>_LDOBJS_DEFAULT Product and library dependant libraries		combined object files with filenames that do not have a suffix, needed to build a specific PROD or LIBRARY for archs with no <name>_LDOBJS_<osclass> specified (deprecated)
<name>_DIR		directory to search for the specified lib. (For libs listed in all PROD_LIBS, LIB_LIBS, <name>_LIBS and USR_LIBS listed below)System libraries do not need a <name>_dir definition.
USR_LIBS		load libraries (e.g. Xt X11) for

(continues on next page)

(continued from previous page)

	all products and libraries
USR_LIBS_<osclass>	os specific load libraries for all makefile links
USR_LIBS_DEFAULT	load libraries for systems with no USR_LIBS_<osclass> specified libs
<name>_LIBS	named prod or library specific ld libraries (e.g. probe_LIBS=X11 Xt)
<name>_LIBS_<osclass>	os-specific libs needed to link named prod or library
<name>_LIBS_DEFAULT	libs needed to link named prod or library for systems with no <name>_LIBS_<osclass> specified
PROD_LIBS	libs needed to link every PROD
PROD_LIBS_<osclass>	os-specific libs needed to link every PROD
PROD_LIBS_DEFAULT	libs needed to link every PROD for archs with no PROD_LIBS_<osclass> specified
LIB_LIBS	libraries to be linked with every library being created
LIB_LIBS_<osclass>	os class specific libraries to be linked with every library being created
LIB_LIBS_DEFAULT	libraries to be linked with every library being created for archs with no LIB_LIBS_<osclass> specified
USR_SYS_LIBS	system libraries (e.g. Xt X11) for all products and libraries
USR_SYS_LIBS_<osclass>	os class specific system libraries for all makefile links
USR_SYS_LIBS_DEFAULT	system libraries for archs with no USR_SYS_LIBS_<osclass>

(continues on next page)

(continued from previous page)

	specified
<name>_SYS_LIBS	named prod or library specific system ld libraries
<name>_SYS_LIBS_<osclass>	os class specific system libs needed to link named prod or library
<name>_SYS_LIBS_DEFAULT	system libs needed to link named prod or library for systems with no <name>_SYS_LIBS_<osclass> specified
PROD_SYS_LIBS	system libs needed to link every PROD
PROD_SYS_LIBS_<osclass>	os class specific system libs needed to link every PROD
PROD_SYS_LIBS_DEFAULT	system libs needed to link every PROD for archs with no PROD_SYS_LIBS_<osclass> specified
LIB_SYS_LIBS	system libraries to be linked with every library being created
LIB_SYS_LIBS_<osclass>	os class specific system libraries to be linked with every library being created
LIB_SYS_LIBS_DEFAULT	system libraries to be linked with every library being created for archs with no LIB_SYS_LIBS_<osclass> specified
SYS_PROD_LIBS	system libs needed to link every PROD for all systems (deprecated)
SYS_PROD_LIBS_<osclass>	os class specific system libs needed to link every PROD (deprecated)
SYS_PROD_LIBS_DEFAULT Compiler flags	system libs needed to link every PROD for systems with no SYS_PROD_LIBS_<osclass> specified (deprecated)

(continues on next page)

(continued from previous page)

USR_CFLAGS	C compiler flags for all systems
USR_CFLAGS_<T_A>	target architecture specific C compiler flags
USR_CFLAGS_<osclass>	os class specific C compiler flags
USR_CFLAGS_DEFAULT	C compiler flags for archs with no USR_CFLAGS_<osclass> specified
<name>_CFLAGS	file specific C compiler flags (e.g. xxxRecord_CFLAGS=-g)
<name>_CFLAGS_<T_A>	file specific C compiler flags for a specific target architecture
<name>_CFLAGS_<osclass>	file specific C compiler flags for a specific os class
USR_CXXFLAGS	C++ compiler flags for all systems (e.g. xyxMain_CFLAGS=-DSDDS)
USR_CXXFLAGS_<T_A>	target architecture specific C++ compiler flags
USR_CXXFLAGS_<osclass>	os-specific C++ compiler flags
USR_CXXFLAGS_DEFAULT	C++ compiler flags for systems with no USR_CXXFLAGS_<osclass> specified
<name>_CXXFLAGS	file specific C++ compiler flags
<name>_CXXFLAGS_<T_A>	file specific C++ compiler flags for a specific target architecture
\ <name>_CXXFLAGS_<osclass>	file specific C++ compiler flags for a specific osclass
USR_CPPFLAGS	C pre-processor flags (for all makefile compiles)
USR_CPPFLAGS_<T_A>	target architecture specific cpp flags

(continues on next page)

(continued from previous page)

USR_CPPFLAGS_<osclass>	os specific cpp flags
USR_CPPFLAGS_DEFAULT	cpp flags for systems with no USR_CPPFLAGS_<osclass> specified
<name>_CPPFLAGS	file specific C pre-processor flags(e.g. xxxRecord_CPPFLAGS=-DDEBUG)
<name>_CPPFLAGS_<T_A>	file specific cpp flags for a specific target architecture
\ <name>_CPPFLAGS_<osclass>	file specific cpp flags for a specific os class
USR_INCLUDES	directories, with -I prefix, to search for include files(e.g. -I$(EPICS_EXTENSIONS_INCLUDE))
USR_INCLUDES_<osclass>	directories, with -I prefix, to search for include files for a specific os class
USR_INCLUDES_DEFAULT	directories, with -I prefix, to search for include files for systems with no <name>_INCLUDES_<osclass> specified
<name>_INCLUDES	directories, with -I prefix, to search for include files when building a specific object file (e.g. -I$(MOTIF_INC))
<name>_INCLUDES_<T_A>	file specific directories, with -I prefix, to search for include files for a specific target architecture
<name>_INCLUDES_<osclass>	file specific directories, with -I prefix, to search for include files for a specific os class
HOST_WARN	Are compiler warning messages desired for host type builds? (YES or NO) (default is YES)
CROSS_WARN	C cross-compiler warning messages desired (YES or NO) (default YES)

(continues on next page)

(continued from previous page)

HOST_OPT	Is host build compiler optimization desired (default is NO optimization)
CROSS_OPT	Is cross-compiler optimization desired (YES or NO) (default is NO optimization)
CMPLR	C compiler selection, TRAD, ANSI or STRICT (default is STRICT)
CXXCMPLR Linker options	C++ compiler selection, NORMAL or STRICT (default is STRICT)
USR_LDFLAGS	linker options (for all makefile links)
USR_LDFLAGS_<osclass>	os specific linker options (for all makefile links)
USR_LDFLAGS_DEFAULT	linker options for systems with no USR_LDFLAGS_<osclass> specified
PROD_LDFLAGS	prod linker options
PROD_LDFLAGS_<osclass>	os specific prod linker options
PROD_LDFLAGS_DEFAULT	prod linker options for systems with no PROD_LDFLAGS_<osclass> specified
LIB_LDFLAGS	library linker options
LIB_LDFLAGS_<osclass>	os specific library linker options
LIB_LDFLAGS_DEFAULT	library linker options for systems with no LIB_LDFLAGS_<osclass> specified
<name>_LDFLAGS	prod or library specific linker options
<name>_LDFLAGS_<osclass>	prod or library specific linker flags for a specific os class
<name>_LDFLAGS_DEFAULT	linker options for systems with no

(continues on next page)

(continued from previous page)

	<name>_LDFLAGS_<osclass> specified
STATIC_BUILD Header files to be installed	Is static build desired (YES or NO) (default is NO). On win32 if STATIC_BUILD=YES then set SHARED_LIBRARIES=NO)
INC	list of include files to install into $(INSTALL_DIR)/include
INC_<osclass>	os specific includes to installed under $(INSTALL_DIR)/include/os/<osclass>
INC_DEFAULT Perl, csh, tcl etc. script installation	include files to install where no INC_<osclass> is specified
SCRIPTS	scripts to install for all systems
SCRIPTS_<osclass>	os-specific scripts to install
SCRIPTS_DEFAULT	scripts to install for systems with no SCRIPTS_<osclass> specified
SCRIPTS_IOC	scripts to install for ioc type archs.
SCRIPTS_IOC_<osclass>	os specific scripts to install for ioc type archs
SCRIPTS_IOC_DEFAULT	scripts to install for ioc type arch systems with no SCRIPTS_IOC_<osclass> specified
SCRIPTS_HOST	scripts to install for host type archs.
SCRIPTS_HOST_<osclass>	os class specific scripts to install for host type archs
SCRIPTS_HOST_DEFAULT	scripts to install for host type arch systems with no OBJS_HOST_<osclass> specified
TCLLIBNAME	list of tcl scripts to install into

(continues on next page)

(continued from previous page)

	$(INSTALL_DIR)/lib/<osclass> (Unix hosts only)
TCLINDEX	name of tcl index file to create from TCLLIBNAME scripts
Object files	The names in the following OBJS definitions should NOT include a suffix (.o or.obj).
OBJS	object files to build and install for all system.
OBJS_<osclass>	os-specific object files to build and install.
OBJS_DEFAULT	object files to build and install for systems with no OBJS_<osclass> specified.
OBJS_IOC	object files to build and install for ioc type archs.
OBJS_IOC_<osclass>	os specific object files to build and install for ioc type archs
OBJS_IOC_DEFAULT	object files to build and install for ioc type arch systems with no OBJS_IOC_<osclass> specified
OBJS_HOST	object files to build and install for host type archs.
OBJS_HOST_<osclass>	os class specific object files to build and install for host type archs
OBJS_HOST_DEFAULT Documentation	object files to build and install for host type arch systems with no OBJS_HOST_<osclass> specified
DOCS	text files to be installed into the $(INSTALL_DIR)/doc directory
HTMLS_DIR	name install Hypertext directory name i.e.

(continues on next page)

(continued from previous page)

	$(INSTALL_DIR)/html/$(HTMLS_DIR)
HTMLS	hypertext files to be installed into the $(INSTALL_DIR)/html/$(HTMLS_DIR) directory
TEMPLATES_DIR	template directory to be created as $(INSTALL_DIR)/templates/$(TEMPLATE_DIR)
TEMPLATES Database Definition files	template files to be installed into $(TEMPLATE_DIR)
DBD	database definition files to be installed or created and installed into $(INSTALL_DBD).
DBDINC	names, without suffix, of menus or record database definitions and headers to be installed or created and installed.
USR_DBDFLAGS	optional flags for dbExpand. Currently only include path (-I <path>) and macro substitution (-S <substitution>) are supported.
DBD_INSTALLS Database Files	files from specified directory to install into $(INSTALL_DBD) (e.g. DBD_INSTALLS = $(APPNAME)/dbd/test.dbd)
DB	database files to be installed or created and installed into $(INSTALL_DB).
DB_INSTALLS	files from specified directory to install into $(INSTALL_DB) (e.g. DB_INSTALLS = $(APPNAME)/db/test.db)
USR_DBFLAGS Options for other programs	optional flags for msi (EPICS Macro Substitution Tool)
YACCOPT	yacc options
LEXOPT	lex options
SNCFLAGS	state notation language, snc, options

(continues on next page)

(continued from previous page)

<name>_SNCFLAGS	product specific state notation language options
E2DB_FLAGS	e2db options
SCH2EDIF_FLAGS	sch2edif options
RANLIBFLAGS	ranlib options
USR_ARFLAGS Facilities for building Java programs	ar options
JAVA	names of Java source files to be built and installed
TESTJAVA	names of Java source files to be built
JAVAINC	names of C header file to be created in O.Common subdirectory
JAR	name of Jar file to be built
JAR_INPUT	names of files to be included in JAR
JAR_MANIFEST	name of manifest file for JAR
USR_JAVACFLAGS	javac tool options
USR_JAVAHFLAGS Facilities for Windows 95/NT resource (.rc) files	javah tool options
RCS	resource files (<name>.rc) needed to build every PROD and LIBRARY
RCS_<osclass>	resource files (<name>.rc) needed to build every PROD and LIBRARY for ioc type archs
RCS_DEFAULT	resource files needed to build every PROD and LIBRARY for ioc type arch systems with no RCS_<osclass> specified
<name>_RCS	resource files needed to build a specific PROD or LIBRARY
<name>_RCS_<osclass>	os specific resource files to

(continues on next page)

(continued from previous page)

	build a specific PROD or LIBRARY
<name>_RCS_DEFAULT Assemblies	resource files needed to build a specific PROD or LIBRARY for ioc type arch systems with no RCS_<osclass> specified
ASSEMBLIES	names of files to be assembled from snippets
COMMON_ASSEMBLIES	names of arch-independent files to be assembled from snippets
<name>_SNIPPETS	snippet files needed to build a specific assembly
<name>_PATTERN Other definitions:	patterns for snippet files (searched from all source directories) needed to build a specific assembly
USR_VPATH	list of directories
BIN_INSTALLS	files from specified directories to be installed into $(INSTALL_BIN) (e.g. BIN_INSTALLS = $(EPICS_BASE_BIN)/aiRecord$(OBJ))
BIN_INSTALLS_<osclass>	os class specific files from specified directories to be installed into $(INSTALL_BIN)
BIN_INSTALLS_DEFAULT	files from specified directories to be installed into $(INSTALL_BIN) for target archs with no BIN_INSTALLS_<osclass> specified
LIB_INSTALLS	files from specified directories to be installed into $(INSTALL_LIB)
LIB_INSTALLS_<osclass>	os class specific files from specified directories to be installed into $(INSTALL_LIB)
LIB_INSTALLS_DEFAULT	files from specified directories to be installed into $(INSTALL_LIB) for target archs with no

(continues on next page)

(continued from previous page)

```
|                                  | LIB_INSTALLS_<osclass>           |
|                                  | specified                        |
+----------------------------------+----------------------------------+
| TARGETS                          | files to create but not install  |
+----------------------------------+----------------------------------+
| INSTALL_LOCATION                 | installation directory (defaults |
|                                  | to $(TOP))                       |
+----------------------------------+----------------------------------+
| GENVERSION                       | If set, the name of a generated  |
|                                  | header file with the module      |
|                                  | version string.                  |
+----------------------------------+----------------------------------+
| GENVERSIONMACRO                  | The CPP macro name written into  |
|                                  | the generated version header     |
|                                  | (default MODULEVERSION).         |
+----------------------------------+----------------------------------+
| GENVERSIONDEFAULT                | The default version string       |
|                                  | written into the generated       |
|                                  | header if no VCS system is in    |
|                                  | use. Leave unset to use build    |
|                                  | time.                            |
+----------------------------------+----------------------------------+
```

Configuration Files

Base Configure Directory

The base/configure directory has the following directory structure:

```
base/

:

configure/

:   os/ tools/
```

Base Configure File Descriptions

The configure files contain definitions and make rules to be included in the various makefiles.

`CONFIG.CrossCommon`

Definitions for all hosts and all targets for a cross build (host different than target).

`CONFIG.gnuCommon`

Definitions for all hosts and all targets for builds using the gnu compiler.

`CONFIG_ADDONS`

Definitions which setup the variables that have and DEFAULT options.

CONFIG_APP_INCLUDE

Definitions to generate include, bin, lib, perl module, db, and dbd directory definitions for RELEASE s.

CONFIG_BASE

EPICS base specific definitions.

CONFIG_BASE_VERSION

Definitions for the version number of EPICS base. This file is used for creating epicsVersion.h which is installed into base/include.

CONFIG_COMMON

Definitions common to all builds.

CONFIG_ENV

Default definitions of the EPICS environment variables. This file is used for creating envData.c which is included in the Com library.

CONFIG_FILE_TYPE

Definitions to allow user created file types.

CONFIG_SITE

File in which you add to or modify make variables in EPICS base. A definition commonly overridden is CROSS_COMPILER_TARGET_ARCHS

CONFIG_SITE_ENV

Defaults for site specific definitions of EPICS environment variables. This file is used for creating envData.c which is included in the Com library.

CONFIG

Include statements for all the other configure files. You can override any definitions in other CONFIG* files by placing override definitions at the end of this file.

RELEASE

Specifies the location of external products such as Tornado II and external such as EPICS base.

RULES

This file just includes the appropriate rules configuration file.

RULES.Db

Rules for building and installing database and database definition files. Databases generated from templates and/or CapFast schematics are supported.

RULES.ioc

Rules which allow building in the iocBoot/ directory of a makeBaseApp created ioc application.

RULES_ARCHS

Definitions and rules which allow building the make target for each target architecture.

RULES_BUILD

Build rules for the Makefiles

RULES_DIRS

Definitions and rules which allow building the make targets in each subdirectory. This file is included by Makefiles in directories with subdirectories to be built.

RULES_EXPAND

Definitions and rules to use expandVars.pl to expand @VAR@ variables in a file.

RULES_FILE_TYPE

Definitions and rules to allow user created CONFIG* and RULES* files and rules to allow user created file types.

RULES_JAVA Definitions and rules which allow building java class files and java jar files. RULES_TARGET

Makefile code to create target specific dependency lines for libraries and product targets.

RULES_TOP

Rules specific to a <top> level directory e.g. uninstall and tar. It also includes the RULES_DIRS file.

Makefile Definitions to allow creation of CONFIG_APP_INCLUDE and installation of the CONFIG* files into the $(INSTALL_LOCATION) directory.

Base configure/os File Descriptions

The configure/os directory contains os specific make definitions. The naming convention for the files in this directory is CONFIG.<host>.<target> where <host> is either the arch for a specific host system or Common for all supported host systems and <target> is either the arch for a specific target system or Common for all supported target systems.

For example, the file CONFIG.Common.vxWorks-pentium will contain make definitions to be used for builds on all host systems when building for a vxWorks-pentium target system.

Also, if a group of host or target files have the same make definitions these common definitions can be moved to a new file which is then included in each host or target file. An example of this is all Unix hosts which have common definitions in a CONFIG.UnixCommon.Common file and all vxWorks targets with definitions in CONFIG.Common.vxWorksCommon.

The base/configure/os directory contains the following os-arch specific definitions

CONFIG.<host>.<target>

Specific host-target build definitions

CONFIG.Common.<target>

Specific target definitions for all hosts

CONFIG.<host>.Common

Specific host definitions for all targets

CONFIG.UnixCommon.Common

Definitions for Unix hosts and all targets

CONFIG.<host>.vxWorksCommon

Specific host definitions for all vx targets

CONFIG_COMPAT

R3.13 arch compatibility definitions

CONFIG_SITE.<host>.<target>

Site specific host-target definitions

CONFIG_SITE.Common.<target>

Site specific target definitions for all hosts

`CONFIG_SITE.<host>.Common`

Site specific host definitions for all targets

Base src/tools File Descriptions

The src/tools directory contains Perl script tools used for the build. They are installed by the build into `$(INSTALL_LOCATION)/bin/$(T_A)` for Host type target archs. The tools currently in this directory are:

convertRelease.pl
: This Perl script does consistency checks for the external <top> definitions in the `RELEASE` file. This script also creates `envPaths`, `cdCommands`, and `dllPath.bat` files for vxWorks and other IOCs.

cvsclean.pl
: This perl script finds and deletes cvs .#* files in all directories of the directory tree.

dos2unix.pl
: This perl script converts text file in DOS CR/LF format to unix ISO format.

expandVars.pl
: This perl tool expands @VAR@ variables while copying a file.

filterWarnings.pl
: This is a perl script that filters compiler warning output (for HP-UX).

fullpathname.pl
: This perl script returns the fullpathname of a file.

installEpics.pl
: This is a Perl script that installs build created files into the install directories.

makeDbDepends.pl
: This perl script searches .substitutions and .template files for entries to create a `DEPENDS` file.

makeIncludeDbd.pl
: This perl script creates an include dbd file from file names

makeMakefile.pl
: This is a perl script that creates a Makefile in the created `O.<arch>` directories.

makeTestfile.pl
: This perl script generates a file `$target.t` which executes a real test program in the same directory.

mkmf.pl
: This perl script generates include file dependencies for targets from source file include statements.

munch.pl
: This is a perl script that creates a `ctdt.c` file for vxWorks target arch builds which lists the c++ static constructors and destructors. See munching in the vxWorks documentation for more information.

replaceVAR.pl
: This is a perl script that changes VAR(xxx) style macros in CapFast generated databases into the `$(xxx)` notation used in EPICS databases.

useManifestTool.pl
: This tools uses MS Visual C++ compiler version number to determine if we want to use the Manifest Tool (status=1) or not (status=0).

Build Documentation Files

Base Documentation Directory

The base/documentation directory contains `README` files to help users setup and build epics/base.

Base Documentation File Descriptions

The files currently in the base/documentation directory are:

README.1st
Instructions for setup and building epics base

README.html
html version of `README.1st`

README.MS_WINDOWS
Microsoft WIN32 specific instructions

README.niCpu030
NI cpu030 specific instructions

README.hpux

```
HPUX 11 (hpux-parisc) specific instructions
```

README.cris
Cris architecture specific instructions

README.tru64unix
Tru64Unix/Alpha specific instructions

README.darwin.html
Installation notes for Mac OS X (Darwin)

BuildingR3.13AppsWithR3.14.html
Describes how to modify a R3.13 vxWorks application so that it builds with release R3.14.1.

ConvertingR3.13AppsToR3.14.html
Describes how to convert a R3.13 vxWorks application so that it contains a R3.14 configure directory and R3.14 Makefiles and builds with R3.14.1.

ConvertingR3.14.0alpha2AppsTobeta1.html
Describes how to modify a R3.14.0alpha1 application so that it builds with release R3.14.0beta1.

ConvertingR3.14.0beta1AppsTobeta2.html
Describes how to modify a R3.14.0beta1 application so that it builds with release R3.14.0beta2.

ConvertingR3.14.0beta2AppsToR3.14.1.html
Describes how to modify a R3.14.0beta2 application so that it builds with release R3.14.1.

ConvertingR3.14.*AppsToR3.14.*.html
Describes how to modify a R3.14.* application so that it builds with next release after R3.14.*.

BuildingR3.13ExtensionsWithR3.14.html
Describes how to modify a R3.13 extension so that it builds with release R3.14.1.

RELEASE_NOTES.html
Describes changes in the R3.14.1 release

KnownProblems.html
List of known problems in EPICS base R3.14.1.

ReleaseChecklist.html
Checklist of things that must be done when creating a new release of EPICS Base.

Startup Files

Base Startup Directory

The base/startup directory contains scripts to help users set the required environment variables and path. The appropriate startup files should be executed before any EPICS builds.

Base Startup File Descriptions

The scripts currently in the base/startup directory are:

EpicsHostArch
c shell script to set `EPICS_HOST_ARCH` environment variable

EpicsHostArch.pl
perl script to set `EPICS_HOST_ARCH` environment variable

Site.profile
Unix bourne shell script to set path and environment variables

Site.cshrc
Unix c shell script to set path and environment variables

cygwin.bat
WIN32 bat file to set path and environment variables for building with cygwin gcc/g++ compilers

win32.bat
WIN32 bat file to set path and environment variables for building with MS Visual C++ compilers

1.13.3 EPICS Process Database Concepts

Tags: beginner user developer

Table of Contents

The EPICS Process Database

An EPICS-based control system contains one or more Input Output Controllers, IOCs. Each IOC loads one or more databases. A database is a collection of records of various types.

A Record is an object with:

- A unique name
- A behavior defined by its type
- Controllable properties (fields)
- Optional associated hardware I/O (device support)

- Links to other records

There are several different types of records available. In addition to the record types that are included in the EPICS base software package, it is possible (although not recommended unless you absolutely need) to create your own record type to perform some specific tasks.

Each record comprises a number of **fields**. Fields can have different functions, typically they are used to configure how the record operates, or to store data items.

Below are short descriptions for the most commonly used record types:

Analog Input and Output (AI and AO) records can store an analog value, and are typically used for things like set-points, temperatures, pressure, flow rates, etc. The records perform number of functions like data conversions, alarm processing, filtering, etc.

Binary Input and Output (BI and BO) records are generally used for commands and statuses to and from equipment. As the name indicates, they store binary values like On/Off, Open/Closed and so on.

Calc and Calcout records can access other records and perform a calculation based on their values. (E.g. calculate the efficiency of a motor by a function of the current and voltage input and output, and converting to a percentage for the operator to read).

Database Functionality Specification

This chapter covers the general functionality that is found in all database records. The topics covered are I/O scanning, I/O address specification, data conversions, alarms, database monitoring, and continuous control:

- *Scanning Specification* describes the various conditions under which a record is processed.
- *Address Specification* explains the source of inputs and the destination of outputs.
- *Conversion Specification* covers data conversions from transducer interfaces to engineering units.
- *Alarm Specification* presents the many alarm detection mechanisms available in the database.
- *Monitor Specification* details the mechanism, which notifies operators about database value changes.
- *Control Specification* explains the features available for achieving continuous control in the database.

These concepts are essential in order to understand how the database interfaces with the process.

The EPICS databases can be created by manual creation of a database "myDatabase.db" text file or using visual tools (VDCT, CapFast). Visual Database Configuration Tool (VDCT), a java application from Cosylab, is a tool for database creation/editing that runs on Linux, Windows, and Sun. The illustrations in this document have been created with VDCT.

Scanning Specification

Scanning determines when a record is processed. A record is *processed* when it performs any actions related to its data. For example, when an output record is processed, it fetches the value which it is to output, converts the value, and then writes that value to the specified location. Each record must specify the scanning method that determines when it will be processed. There are three scanning methods for database records:

(1) periodic,

(2) event, and

(3) passive.

Periodic scanning occurs on set time intervals.

Event scanning occurs on either an I/O interrupt event or a user-defined event.

Passive scanning occurs when the records linked to the passive record are scanned, or when a value is "put" into a passive record through the database access routines.

For periodic or event scanning, the user can also control the order in which a set of records is processed by using the PHASE mechanism. The number in the

PHAS field allows to define the relative order in which records are processed within a scan cycle:

- Records with PHAS=0 are processed first
- Then those with PHAS=1, PHAS=2, etc.

For event scanning, the user can control the priority at which a record will process. The PRIO field selects Low/Medium/High priority for Soft event and I/O Interrupts.

In addition to the scan and the phase mechanisms, there are data links and forward processing links that can be used to cause processing in other records.

Periodic Scanning

The periodic scan tasks run as close as possible to the specified frequency. When each periodic scan task starts, it calls the gettime routine, then processes all of the records on this period. After the processing, gettime is called again and this thread sleeps the difference between the scan period and the time to process the records. For example, if it takes 100 milliseconds to process all records with "1 second" scan period, then the 1 second scan period will start again 900 milliseconds after completion. The following periods for scanning database records are available by default, though EPICS can be configured to recognize more scan periods:

- 10 second
- 5 second
- 2 second
- 1 second
- .5 second
- .2 second
- .1 second

The period that best fits the nature of the signal should be specified. A five-second interval is adequate for the temperature of a mass of water because it does not change rapidly. However, some power levels may change very rapidly, so they need to be scanned every 0.5 seconds. In the case of a continuous control loop, where the process variable being controlled can change quickly, the 0.1 second interval may be the best choice.

For a record to scan periodically, a valid choice must be entered in its SCAN field. Actually, the available choices depend on the configuration of the menuScan.dbd file. As with most other fields which consists of a menu of choices, the choices available for the SCAN field can be changed by editing the appropriate .dbd (database definition) file. dbd files are ASCII files that are used to generate header files that are, in turn, are used to compile the database code. Many dbd files can be used to configure other things besides the choices of menu fields.

Here is an example of a menuScan.dbd file, which has the default menu choices for all periods listed above as well as choices for event scanning, passive scanning, and I/O interrupt scanning:

```
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")
}
```

The first three choices must appear first and in the order shown. The remaining definitions are for the periodic scan rates, which must appear in the order slowest to fastest (the order directly controls the thread priority assigned to the particular scan rate, and faster scan rates should be assigned higher thread priorities). At IOC initialization, the menu choice strings are read at scan initialization. The number of periodic scan rates and the period of each rate is determined from the menu choice strings. Thus the periodic scan rates can be changed by changing menuScan.dbd and loading this version via dbLoadDatabase. The only requirement is that each periodic choice string must begin with a number and be followed by any of the following unit strings:

- second or seconds
- minute or minutes
- hour or hours
- Hz or Hertz

For example, to add a choice for 0.015 seconds, add the following line after the 0.1 second choice:

```
choice(menuScan_015_second, " .015 second")
```

The range of values for scan periods can be from one clock tick to the maximum number of ticks available on the system (for example, vxWorks out of the box supports 0.015 seconds or a maximum frequency of 60 Hz). Note, however, that the order of the choices is essential. The first three choices must appear in the above order. Then the remaining choices should follow in descending order, the biggest time period first and the smallest last.

Event Scanning

There are two types of events supported in the input/output controller (IOC) database, the I/O interrupt event and the user-defined event. For each type of event, the user can specify the scheduling priority of the event using the PRIO or priority field. The scheduling priority refers to the priority the event has on the stack relative to other running tasks. There are three possible choices: LOW, MEDIUM, or HIGH. A low priority event has a priority a little higher than Channel Access. A medium priority event has a priority about equal to the median of periodic scanning tasks. A high priority event has a priority equal to the event scanning task.

I/O Interrupt Events

Scanning on I/O interrupt causes a record to be processed when a driver posts an I/O Event. In many cases these events are posted in the interrupt service routine. For example, if an analog input record gets its value from an I/O card and it specifies I/O interrupt as its scanning routine, then the record will be processed each time the card generates an interrupt (not all types of I/O cards can generate interrupts). Note that even though some cards cannot actually generate interrupts, some driver support modules can simulate interrupts. In order for a record to scan on I/O interrupts, its SCAN field must specify I/O Intr.

User-defined Events

The user-defined event mechanism processes records that are meaningful only under specific circumstances. User-defined events can be generated by the post_event() database access routine. Two records, the event record and the timer record, are also used to post events. For example, there is the timing output, generated when the process is in a state where a control can be safely changed. Timing outputs are controlled through Timer records, which have the ability to generate interrupts. Consider a case where the timer record is scanned on I/O interrupt and the timer record's event field (EVNT) contains an event number. When the record is scanned, the user-defined event will be posted. When the event is posted, all records will be processed whose SCAN field specifies event and whose event number is the same as the generated event. User-defined events can also be generated through software. Event numbers are configurable and should be controlled through the project engineer. They only need to be unique per IOC because they only trigger processing for records in the same IOC.

All records that use the user-defined event mechanism must specify Event in their SCAN field and an event number in their EVNT field.

Passive Scanning

Passive records are processed when they are referenced by other records through their link fields or when a channel access put is done to them.

Channel Access Puts to Passive Scanned Records

In this case where a channel access put is done to a record, the field being written has an attribute that determines if this put causes record processing. In the case of all records, putting to the VAL field causes record processing. Consider a binary output that has a SCAN of Passive. If an operator display has a button on the VAL field, every time the button is pressed, a channel access put is sent to the record. When the VAL field is written, the Passive record is processed and the specified device support is called to write the newly converted RVAL to the device specified in the OUT field through the device support specified by DTYP. Fields determined to change the way a record behaves, typical cause the record to process. Another field that would cause the binary output to process would be the ZSV; which is the alarm severity if the binary output record is in state Zero (0). If the record was in state 0 and the severity of being in that state changed from No Alarm to Minor Alarm, the only way to catch this on a SCAN Passive record is to process it. Fields are configured to cause binary output records to process in the bo.dbd file. The ZSV severity is configured as follows:

```
field(ZSV,DBF_MENU) {
  prompt("Zero Error Severity")
  promptgroup(GUI_ALARMS)
  pp(TRUE)
  interest(1)
  menu(menuAlarmSevr)
}
```

where the line "pp(TRUE)" is the indication that this record is processed when a channel access put is done.

Database Links to Passive Record

The records in the process database use link fields to configure data passing and scheduling (or processing). These fields are either INLINK, OUTLINK, or FWDLINK fields.

Forward Links

In the database definition file (.dbd) these fields are defined as follows:

```
field(FLNK,DBF_FWDLINK) {
  prompt("Forward Process Link")
  promptgroup(GUI_LINKS)
  interest(1)
}
```

If the record that is referenced by the FLNK field has a SCAN field set to "Passive", then the record is processed after the record with the FLNK. The FLNK field only causes record processing, no data is passed. In (*Figure 1*), three records are shown. The ai record "Input_2" is processed periodically. At each interval, Input_2 is processed. After Input_2 has read the new input, converted it to engineering units, checked the alarm condition, and posted monitors to Channel Access, then the calc record "Calculation_2" is processed. Calculation_2 reads the input, performs the calculation, checked the alarm condition, and posted monitors to Channel Access, then the ao record "Output_2" is processed. Output_2 reads the desired output, rate limits it, clamps the range, calls the device support for the OUT field, checks alarms, posts monitors and then is complete.

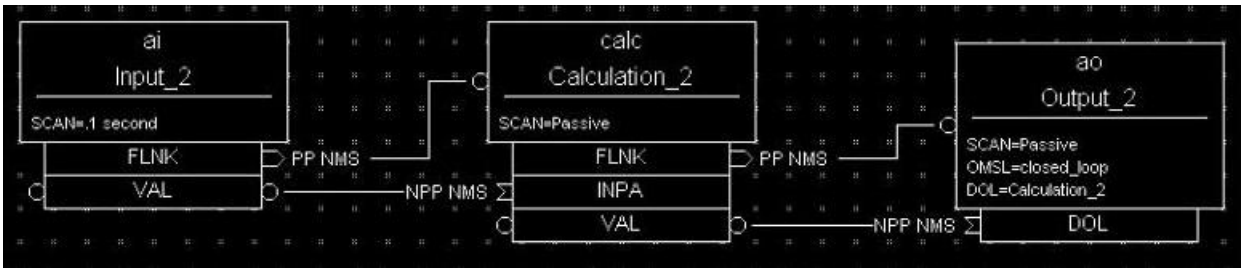


Figure 1. Input Links

Input links normally fetch data from one field into a field in the referring record. For instance, if the INPA field of a CALC record is set to Input_3.VAL, then the VAL field is fetched from the Input_3 record and placed in the A field of the CALC record. These data links have an attribute that specify if a passive record should be processed before the value is returned. The default for this attribute is NPP (no process passive). In this case, the record takes the VAL field and returns it. If they are set to PP (process passive), then the record is processed before the field is returned.

In *Figure 2*), the PP attribute is used. In this example, Output_3 is processed periodically. Record processing first fetching the DOL field. As the DOL field has the PP attribute set, before the VAL field of Calc_3 is returned, the record is processed. The first thing done by the ai record Input_3 does is to read the input. It then converts the RVAL field to engineering units and places this in the VAL field, checks alarms, posts monitors, and then returns. The calc record then fetches the VAL field field from Input_3, places it in the A field, computes the calculation, checks alarms, posts monitors, the returns. The ao record, Output_3, then fetches the VAL field from the CALC record, applies rate of change and limits, write the new value, checks alarms, posts monitors and completes.

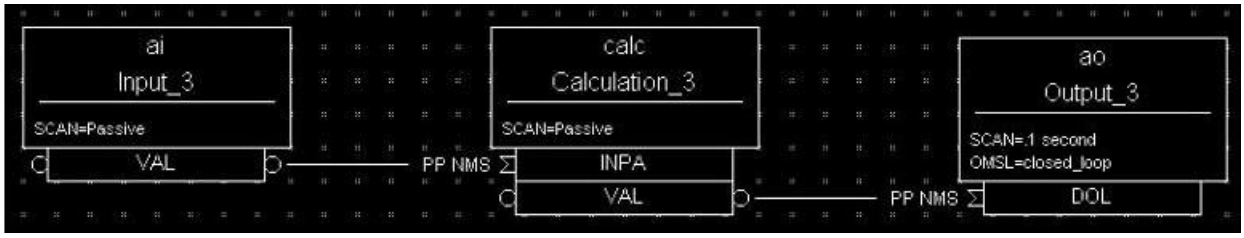


Figure 2

In *Figure 3*) the PP/NPP attribute is used to calculate a rate of change. At 1 Hz, the calculation record is processed. It fetches the inputs for the calc record in order. As INPA has an attribute of NPP, the VAL field is taken from the ai record. Before INPB takes the VAL field from the ai record it is processed, as the attribute on this link is PP. The new ai value is placed in the B field of the calc record. A-B is the VAL field of the ai one second ago and the current VAL field.

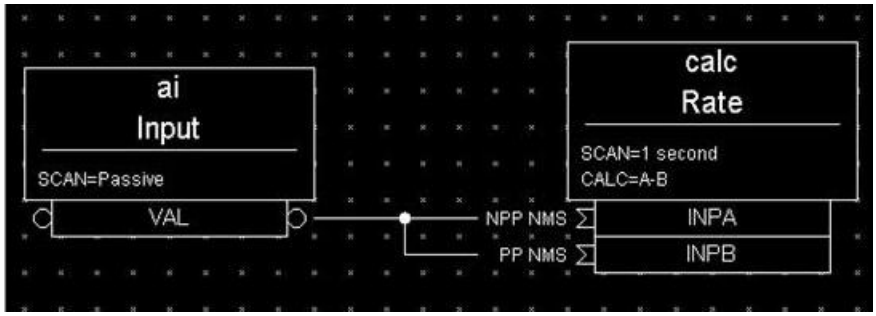


Figure 3

Process Chains

Links can be used to create complex scanning logic. In the forward link example above, the chain of records is determined by the scan rate of the input record. In the PP example, the scan rate of the chain is determined by the rate of the output. Either of these may be appropriate depending on the hardware and process limitations.

Care must be taken as this flexibility can also lead to some incorrect configurations. In these next examples we look at some mistakes that can occur.

In *Figure 4*) two records that are scanned at 10 Hz make references to the same Passive record. In this case, no alarm or error is generated. The Passive record is scanned twice at 10 Hz. The time between the two scans depends on what records are processed between the two periodic records.

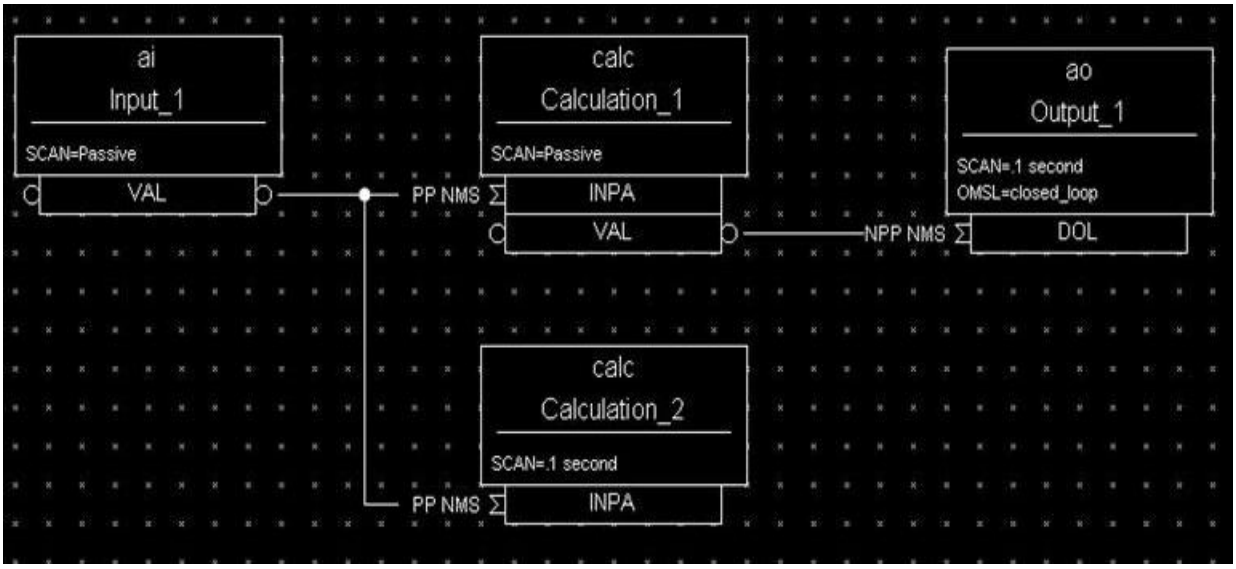


Figure 4

In *Figure 5*), several circular references are made. As the record processing is recursively called for links, the record containing the link is marked as active during the entire time that the chain is being processed. When one of these circular references is encountered, the active flag is recognized and the request to process the record is ignored.

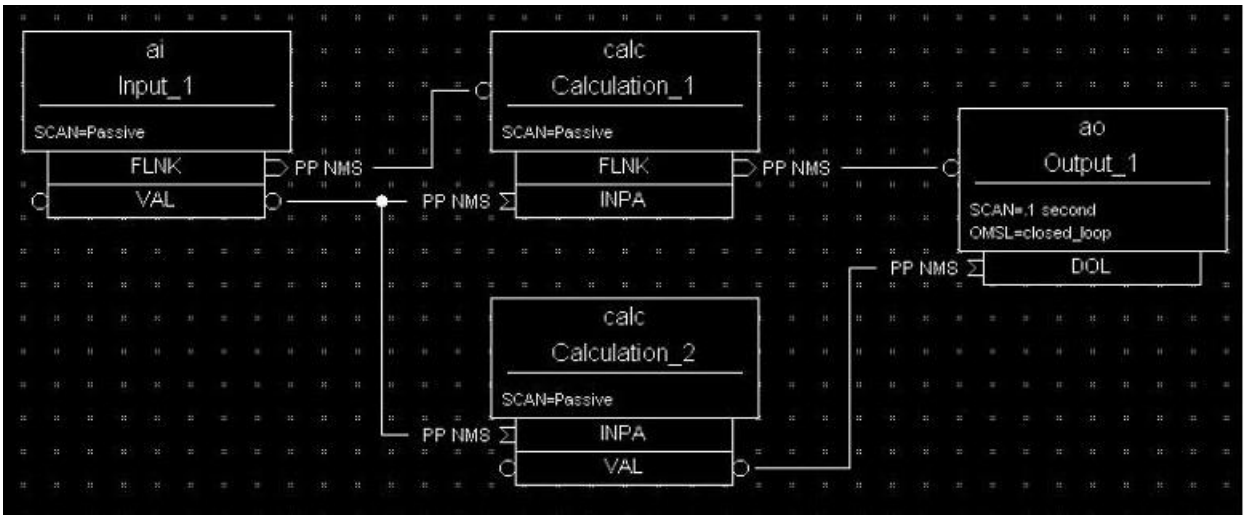


Figure 5

Channel Access Links

A Channel Access link is an input link or output link that specifies a link to a record located in another IOC or an input and output link with one of the following attributes: CA, CP, or CPP.

Channel Access Input Links

If the input link specifies CA, CP, or CPP, regardless of the location of the process variable being referenced, it will be forced to be a Channel Access link. This is helpful for separating process chains that are not tightly related. If the input link specifies CP, it also causes the record containing the input link to process whenever a monitor is posted, no matter what the record's SCAN field specifies. If the input link specifies CPP, it causes the record to be processed if and only if the record with the CPP link has a SCAN field set to Passive. In other words, CP and CPP cause the record containing the link to be processed with the process variable that they reference changes.

Channel Access Output Links

Only CA is appropriate for an output link. The write to a field over channel access causes processing as specified in *Channel Access Puts to Passive Scanned Records*.

Channel Access Forward Links

Forward links can also be Channel Access links, either when they specify a record located in another IOC or when they specify the CA attributes. However, forward links will only be made Channel Access links if they specify the PROC field of another record.

Maximize Severity Attribute

The Maximize Severity attribute is one of the following :

- NMS (Non-Maximize Severity)
- MS (Maximize Severity)
- MSS (Maximize Status and Severity)
- MSI (Maximize Severity if Invalid)

It determines whether alarm severity is propagated across links. If the attribute is MSI only a severity of INVALID_ALARM is propagated; settings of MS or MSS propagate all alarms that are more severe than the record's current severity. For input links the alarm severity of the record referred to by the link is propagated to the record containing the link. For output links the alarm severity of the record containing the link is propagated to the record referred to by the link. If the severity is changed the associated alarm status is set to LINK_ALARM, except if the attribute is MSS when the alarm status will be copied along with the severity.

The method of determining if the alarm status and severity should be changed is called ``maximize severity''. In addition to its actual status and severity, each record also has a new status and severity. The new status and severity are initially 0, which means NO_ALARM. Every time a software component wants to modify the status and severity, it first checks the new severity and only makes a change if the severity it wants to set is greater than the current new severity. If it does make a change, it changes the new status and new severity, not the current status and severity. When database monitors are checked, which is normally done by a record processing routine, the current status and severity are set equal to the new values and the new values reset to zero. The end result is that the current alarm status and severity reflect the highest severity outstanding alarm. If multiple alarms of the same severity are present the alarm status reflects the first one detected.

Phase

The PHAS field is used to order the processing of records that are scanned at the same time, i.e., records that are scanned periodically at the same interval and priority, or that are scanned on the same event. In this manner records dependent upon other records can be assured of using current data.

To illustrate this we will look at an example from the previous section, with the records, however, being scanned periodically instead of passively (*Figure 6*). In this example each of these records specifies .1 second; thus, the records are synchronous. The phase sequence is used to assure that the analog input is processed first, meaning that it fetches its value from the specified location and places it in the VAL field (after any conversions). Next, the calc record will be processed, retrieving its value from the analog input and performing its calculation. Lastly, the analog output will be processed, retrieving its desired output value from the calc record's VAL field (the VAL field contains the result of the calc record's calculations) and writing that value to the location specified it its OUT link. In order for this to occur, the PHAS field of the analog input record must specify 0, the PHAS field of the calculation record must specify 1, and the analog output's PHAS field must specify 2.

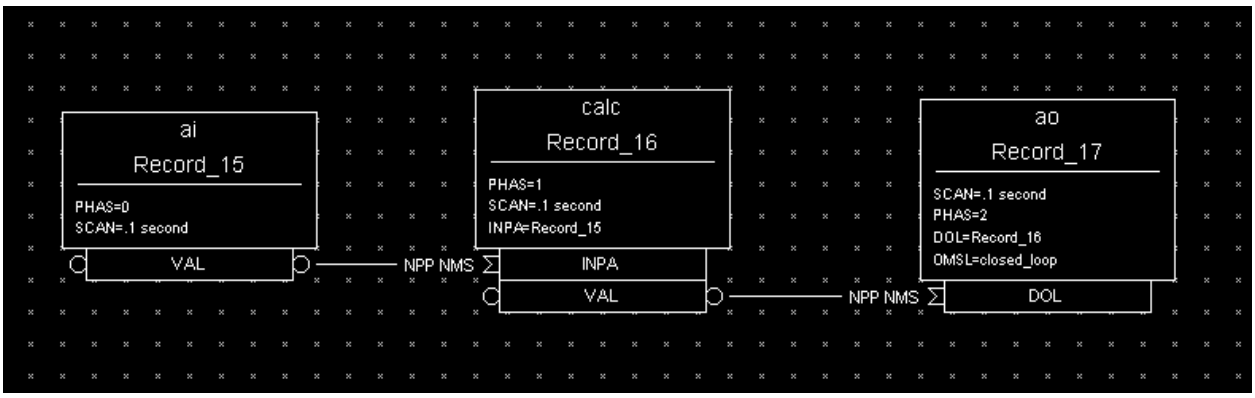


Figure 6

It is important to understand that in the above example, no record causes another to be processed. The phase mechanism instead causes each to process in sequence.

PVAccess Links

When built against Base >= 3.16.1, support is enabled for PVAccess links, which are analogous to Channel Access (CA) links. However, the syntax for PVA links is quite different.

The authoritative documentation is available in the git repository, pva2pva.

Note: The "dbjlr" and "dbpvar" IOC shell command provide information about PVA links in a running IOC.

A simple configuration using defaults is

```
record(longin, "tgt") {}
record(longin, "src") {
  field(INP, {pva:"tgt"})
}
```

This is a shorthand for

```
record(longin, "tgt") {}
record(longin, "src") {
    field(INP, {pva:{pv:"tgt"}})
}
```

Some additional keys (beyond “pv”) may be used. Defaults are shown in the example below:

```
record(longin, "tgt") {}
record(longin, "src") {
  field(INP, {pva:{
    pv:"tgt",
    field:"", # may be a sub-field
    local:false,# Require local PV
    Q:4, # monitor queue depth
    pipeline:false, # require that server uses monitor
    # flow control protocol
    proc:none, # Request record processing
    #(side-effects).
    sevr:false, # Maximize severity.
    time:false, # set record time during getValue
    monorder:0, # Order of record processing as a result #of CP and CPP
    retry:false,# allow Put while disconnected.
    always:false,# CP/CPP input link process even when # .value field hasn't changed
    defer:false # Defer put
  }})
}
```

pv: Target PV name

The PV name to search for. This is the same name which could be used with ‘pvget’ or other client tools.

field: Structure field name

The name of a sub-field of the remotely provided Structure. By default, an empty string “” uses the top-level Structure.

If the top level structure, or a sub-structure is selected, then it is expeccted to conform to NTScalar, NTScalarArray, or NTEnum to extract value and meta-data.

If the sub-field is an PVScalar or PVScalarArray, then a value will be taken from it, but not meta-data will be available.

local: Require local PV

When true, link will not connect unless the named PV is provided by the local (QSRV) data provider.

Q: Monitor queue depth

Requests a certain monitor queue depth. The server may, or may not, take this into consideration when selecting a queue depth.

pipeline: Monitor flow control

Expect that the server supports PVA monitor flow control. If not, then the subscription will stall (ick.)

proc: Request record processing (side-effects)

The meaning of this option depends on the direction of the link.

For output links, this option allows a request for remote processing (side-effects).

- none (default) - Make no special request. Uses a server specific default.
- false, “NPP” - Request to skip processing.
- true, “PP” - Request to force processing.
- “CP”, “CPP” - For output links, an alias for “PP”.

For input links, this option controls whether the record containing the PVA link will be processed when subscription events are received.

- none (default), false, “NPP” - Do not process on subscription updates.
- true, “CP” - Always process on subscription updates.
- “PP”, “CPP” - Process on subscription updates if SCAN=Passive

sevr: Alarm propagation

This option controls whether reading a value from an input PVA link has the addition effect of propagating any alarm via the Maximize Severity process.

- false - Do not maximize severity.
- true - Maximize alarm severity
- “MSI” - Maximize only if the remote severity is INVALID.

time: Time propagation

Somewhat analogous to sevr: applied to timestamp. When true, the record TIME field is updated when the link value is read.

Warning

TSEL must be set to -2 for time:true to have an effect.

monorder: Monitor processing order

When multiple record target the same target PV, and request processing on subscription updates. This option allows the order of processing to be specified.

Record are processed in increasing order. monorder=-1 is processed before monorder=0. Both are processed before monorder=1.

defer: Defer put

By default (defer=false) an output link will immediately start a PVA Put operation. defer=true will store the new value in an internal cache, but not start a PVA Put.

This option, in combination with field: allows a single Put to contain updates to multiple sub-fields.

retry: Put while disconnected

Allow a Put operation to be queued while the link is disconnected. The Put will be executed when the link becomes connected.

always: CP/CPP always process

By default (always:false) a subscription update will only cause a CP input link to scan if the structure field (cf. field: option) is marked as changed. Set to true to override this, and always process the link.

Link semantics/behavior

This section attempts to answer some questions about how links behave in certain situations.

Links are evaluated in three basic contexts.

- dbPutLink()/dbScanFwdLink()
- dbGetLink() of non-CP link
- dbGetLink() during a scan resulting from a CP link.

An input link can bring in a Value as well as meta-data, alarm, time, and display/control info. For input links, the PVA link engine attempts to always maintain consistency between Value, alarm, and time. However, consistency between these, and the display/control info is only ensured during a CP scan.

Address Specification

Address parameters specify where an input record obtains input, where an output record obtains its desired output values, and where an output record writes its output. They are used to identify links between records, and to specify the location of hardware devices. The most common link fields are OUT, an output link, INP, an input link, and DOL (desired output location), also an input link.

There are three basic types of address specifications, which can appear in these fields: hardware addresses, database addresses, and constants. Note that not all links support all three types, though some do. However, this doesn't hold true for algorithmic records, which cannot specify hardware addresses. Algorithm records are records like the Calculation, PID, and Select records. These records are used to process values retrieved from other records. Consult the documentation for each record.

Hardware Addresses

The interface between EPICS process database logic and hardware drivers is indicated in two fields of records that support hardware interfaces: DTYP and INP/OUT. The DTYP field is the name of the device support entry table that is used to interface to the device. The address specification is dictated by the device support. Some conventions exist for several buses that are listed below. Lately, more devices have just opted to use a string that is then parsed by the device support as desired. This specification type is called INST I/O. The other conventions listed here include: VME, Allen-Bradley, CAMAC, GPIB, BITBUS, VXI, and RF. The input specification for each of these is different. The specification of these strings must be acquired from the device support code or document.

INST

The INST I/O specification is a string that is parsed by the device support. The format of this string is determined by the device support.

@parm

For INST I/O

- @ precedes optional string *parm*

VME Bus

The VME address specification format differs between the various devices. In all of these specifications the '#' character designates a hardware address. The three formats are:

#C*x* S*y* *@parm*

For analog in, analog out, and timer

- C precedes the card number *x*
- S precedes the signal number *y*
- @ precedes optional string *parm*

The card number in the VME addresses refers to the logical card number. Card numbers are assigned by address convention; their position in the backplane is of no consequence. The addresses are assigned by the technician who populates the backplane, with the logical numbers welldocumented. The logical card numbers start with 0 as do the signal numbers. *parm* refers to an arbitrary string of up to 31 characters and is device specific.

Allen-Bradley Bus

The Allen-Bradley address specification is a bit more complicated as it has several more fields. The '#' designates a hardware address. The format is:

#L*a* A*b* C*c* S*d* *@parm'*

All record types

- L precedes the serial link number *a* and is optional - default 0
- A precedes the adapter number *b* and is optional - default 0
- C precedes the card number *c*
- S precedes the signal number *d*
- @ precedes optional string *parm*

The card number for Allen-Bradley I/O refers to the physical slot number, where 0 is the slot directly to the right of the adapter card. The AllenBradley I/O has 12 slots available for I/O cards numbered 0 through 11. Allen-Bradley I/O may use double slot addresses which means that slots 0,2,4,6,8, and 10 are used for input modules and slots 1,3,5,7,9 and 11 are used for output modules. It's required to use the double slot addressing mode when the 1771IL card is used as it only works in double slot addressing mode. This card is required as it provides Kilovolt isolation.

Camac Bus

The CAMAC address specification is similar to the Allen-Bradley address specification. The '#' signifies a hardware address. The format is:

#B*a* C*b* N*c* A*d* F*e* @*parm*

For waveform digitizers

- B precedes the branch number *a* C precedes the crate number *b*
- N precedes the station number *c*
- A precedes the subaddress *d* (optional)
- F precedes the function *e* (optional)
- @ precedes optional string *parm*

The waveform digitizer supported is only one channel per card; no channel was necessary.

Others

The GPIB, BITBUS, RF, and VXI card-types have been added to the supported I/O cards. A brief description of the address format for each follows. For a further explanation, see the specific documentation on each card.

#L*a* A*b* @*parm*
For GPIB I/O

- L precedes the link number *a*
- A precedes the GPIB address *b*
- @ precedes optional string *parm*

#L*a* N*b* P*c* S*d* @*parm*

For BITBUS I/O

- L precedes the link *a*, i.e., the VME bitbus interface
- N precedes the bitbus node *b*
- P precedes the port on node *c*
- S precedes the signal on port *d*
- @ precedes optional string *parm*

#V*a* C*b* S*c* @*parm*

For VXI I/O, dynamic addressing

- V precedes the VXI frame number *a*
- C precedes the slot within VXI frame *b*
- S precedes the signal number *c*

- @ precedes optional string *parm*

#V*a* S*b* @*parm*

For VXI I/O, static addressing

- V precedes the logical address *a*
- S precedes the signal number *b*
- @ precedes optional string *parm*

Database Addresses

Database addresses are used to specify input links, desired output links, output links, and forward processing links. The format in each case is the same:

<RecordName>.<FieldName>

where RecordName is simply the name of the record being referenced, '.' is the separator between the record name and the field name, and FieldName is the name of the field within the record.

The record name and field name specification are case sensitive. The record name can be a mix of the following: a-z A-Z 0-9 _ - : . [ ] < > ;. The field name is always upper case. If no field name is specified as part of an address, the value field (VAL) of the record is assumed. Forward processing links do not need to include the field name because no value is returned when a forward processing link is used; therefore, a forward processing link need only specify a record name.

Basic typecast conversions are made automatically when a value is retrieved from another record–integers are converted to floating point numbers and floating point numbers are converted to integers. For example, a calculation record which uses the value field of a binary input will get a floating point 1 or 0 to use in the calculation, because a calculation record's value fields are floating point numbers. If the value of the calculation record is used as the desired output of a multi-bit binary output, the floating point result is converted to an integer, because multi-bit binary outputs use integers.

Records that use soft device support routines or have no hardware device support routines are called *soft records*. See the chapter on each record for information about that record's device support.

Constants

Input link fields and desired output location fields can specify a constant instead of a hardware or database address. A constant, which is not really an address, can be an integer value in whatever format (hex, decimal, etc.) or a floating-point value. The value field is initialized to the constant when the database is initialized, and at run-time the value field can be changed by a database access routine. For instance, a constant may be used in an input link of a calculation record. For non-constant links, the calc record retrieves the values from the input links, and places them in a corresponding value field. For constant links, the value fields are initialized with the constant, and the values can be changed by modifying the value field, not the link field. Thus, because the calc record uses its value fields as the operands of its expression, the constant becomes part of the calculation.

When nothing is specified in a link field, it is a NULL link. Before Release 3.13, the value fields associated with the NULL link were initialized with the value of zero. From Release 3.13 onwards, the value fields associated with the links are not initialized.

A constant may also be used in the desired output location or DOL field of an output record. In such a case, the initial desired output value (VAL) will be that constant. Any specified conversions are performed on the value before it is written as long as the device support module supports conversions (the Soft Channel device support routine does not perform conversions). The desired output value can be changed by an operator at run-time by writing to the value field.

A constant can be used in an output link field, but no output will be written if this is the case. Be aware that this is not considered an error by the database checking utilities.

Conversion Specification

Conversion parameters are used to convert transducer data into meaningful data. Discrete signals require converting between levels and states (i.e., on, off, high, low, etc.). Analog conversions require converting between levels and engineering units (i.e., pressure, temperature, level, etc.). These conversions are made to provide operators and application codes with values in meaningful units.

The following sections discuss these types of conversions. The actual field names appear in capital letters.

Discrete Conversions

The most simple type of discrete conversion would be the case of a discrete input that indicates the on/off state of a device. If the level is high it indicates that the state of the device is on. Conversely, if the level is low it indicates that the device is off. In the database, parameters are available to enter strings which correspond to each level, which, in turn, correspond to a state (0,1). By defining these strings, the operator is not required to know that a specific transducer is on when the level of its transmitter is high or off when the level is low. In a typical example, the conversion parameters for a discrete input would be entered as follows:

Zero Name (ZNAM): Off
One Name (ONAM): On

The equivalent discrete output example would be an on/off controller. Let's consider a case where the safe state of a device is On, the zero state. The level being low drives the device on, so that a broken cable will drive the device to a safe state. In this example the database parameters are entered as follows:

Zero Name (ZNAM): On
One Name (ONAM): Off

By giving the outside world the device state, the information is clear. Binary inputs and binary outputs are used to represent such on/off devices.

A more complex example involving discrete values is a multi-bit binary output record. Consider a two state valve which has four states-Traveling, full open, full closed, and disconnected. The bit pattern for each control state is entered into the database with the string that describes that state. The database parameters for the monitor would be entered as follows:

Number of Bits (NOBT): 2
First Input Bit Spec (INP): Address of the least significant bit
Zero Value (ZRVL): 0
One Value (ONVL): 1
Two Value (TWVL): 2
Three Value (THVL): 3
Zero String (ZRST): Traveling
One String (ONST): Open
Two String (TWST): Closed
Three String (THST): Disconnected

In this case, when the database record is scanned, the monitor bits are read and compared with the bit patterns for each state. When the bit pattern is found, the device is set to that state. For instance, if the two monitor bits read equal 10 (binary), the Two value is the corresponding value, and the device would be set to state 2 which indicates that the valve is Closed.

If the bit pattern is not found, the device is in an unknown state. In this example all possible states are defined.

In addition, the DOL fields of binary output records (bo and mbbo) will accept values in strings. When they retrieve the string or when the value field is given a string via put_enum_strs, a match is sought with one of the states. If a match is found, the value for that state is written.

Analog Conversions

Analog conversions require knowledge of the transducer, the filters, and the I/O cards. Together they measure the process, transmit the data, and interface the data to the IOC. Smoothing is available to filter noisy signals. The smoothing argument is a constant between 0 and 1 and is specified in the SMOO field. It is applied to the converted hardware signal as follows:

eng units = (new eng units × (1 - smoothing)) + (old eng units × smoothing)

The analog conversions from raw values to engineering units can be either linear or breakpoint conversions.

Whether an analog record performs linear conversions, breakpoint conversions, or no conversions at all depends on how the record's LINR field is configured. The possible choices for the LINR field are as follows:

- LINEAR
- SLOPE
- NO CONVERSION
- typeKdegF
- typeKdegC
- typeJdegF
- typeJdegC

If either LINEAR or SLOPE is chosen, the record performs a linear conversion on the data. If NO CONVERSION is chosen, the record performs no conversion on its data. The other choices are the names of breakpoint tables. When one of these is specified in the LINR field, the record uses the specified table to convert its data.

Note: Additional breakpoint tables are often added at specific sites, so more breakpoint tables than are listed here may be available at the user's site.

The following sections explain linear and breakpoint conversions.

Linear Conversions

The engineering units full scale and low scale are specified in the EGUF and EGUL fields, respectively. The values of the EGUF and EGUL fields correspond to the maximum and minimum values of the transducer, respectively. Thus, the value of these fields is device dependent. For example, if the transducer has a range of -10 to +10 volts, then the EGUF field should be 10 and the EGUL field should be -10. In all cases, the EGU field is a string that contains the text to indicate the units of the value.

The distinction between the LINEAR and SLOPE settings for the LINR field are in how the conversion parameters are calculated:

With LINEAR conversion the user must set EGUL and EGUF to the lowest and highest possible engineering units values respectively that can be converted by the hardware. The device support knows the range of the raw data and calculates ESLO and EOFF from them.

SLOPE conversion requires the user to calculate the appropriate scaling and offset factors and put them directly in ESLO and EOFF.

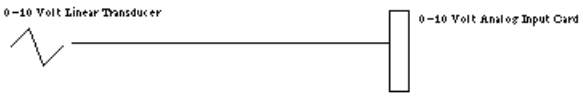
There are three formulas to know when considering the linear conversion parameters. The conversion from measured value to engineering units is as follows:

$$\text{engunits} = \text{eng units low} + \frac{\text{measured A/D counts}}{\text{full scale A/D counts}} * (\text{eng units full scale - eng units low})$$

In the following examples the determination of engineering units full scale and low scale is shown. The conversion to engineering units is also shown to familiarize the reader with the signal conversions from signal source to database engineering units.

Transducer Matches the I/O module

First let us consider a linear conversion. In this example, the transducer transmits 0-10 Volts, there is no amplification, and the I/O card uses a 0-10 Volt interface.



The transducer transmits pressure: 0 PSI at 0 Volts and 175 PSI at 10 Volts. The engineering units full scale and low scale are determined as follows:

eng. units full scale = 17.5 × 10.0
eng. units low scale = 17.5 × 0.0

The field entries in an analog input record to convert this pressure will be as follows:

LINR: Linear
EGUF: 175.0
EGUL: 0
EGU: PSI

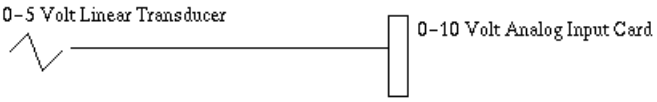
The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = 0 + \frac{\text{measured A/D counts}}{4095} * (175 - 0)$$

When the pressure is 175 PSI, 10 Volts is sent to the I/O module. At 10 Volts the signal is read as 4095. When this is plugged into the conversion, the value is 175 PSI.

Transducer Lower than the I/O module

Let's consider a variation of this linear conversion where the transducer is 0-5 Volts.



In this example the transducer is producing 0 Volts at 0 PSI and 5 Volts at 175 PSI. The engineering units full scale and low scale are determined as follows:

eng. units low scale = 35 × 10 eng. units full scale = 35 × 0

The field entries in an analog record to convert this pressure will be as follows:

LINR: Linear
EGUF: 350
EGUL: 0
EGU: PSI

The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = 0 + \frac{\text{measured A/D counts}}{4095} * (350 - 0)$$

Notice that at full scale the transducer will generate 5 Volts to represent 175 PSI. This is only half of what the input card accepts; input is 2048.

Let's plug in the numbers to see the result:

$$0 + (2048/4095) * (350 - 0) = 175$$

In this example we had to adjust the engineering units full scale to compensate for the difference between the transmitter and the analog input card.

Transducer Positive and I/O module bipolar

Let's consider another variation of this linear conversion where the input card accepts -10 Volts to 10 Volts (i.e. Bipolar instead of Unipolar).

In this example the transducer is producing 0 Volts at 0 PSI and 10 Volts at 175 PSI. The input module has a different range of voltages and the engineering units full scale and low scale are determined as follows:

eng. units full scale = 17.5 × 10 eng. units low scale = 17.5 × (-10)

The database entries to convert this pressure will be as follows:

LINR: Linear
EGUF: 175
EGUL: -175
EGU: PSI

The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = -175 + \frac{\text{measured A/D counts}}{4095} * (175 - (-175))$$

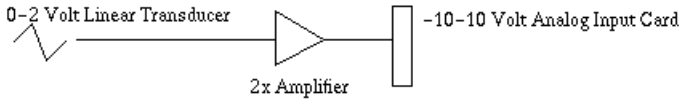
Notice that at low scale the transducer will generate 0 Volts to represent 0 PSI. Because this is half of what the input card accepts, it is input as 2048. Let's plug in the numbers to see the result:

$$-175 + (2048/4095) * (175 - (-175)) = 0$$

In this example we had to adjust the engineering units low scale to compensate for the difference between the unipolar transmitter and the bipolar analog input card.

Combining Linear Conversion with an Amplifier

Let's consider another variation of this linear conversion where the input card accepts -10 Volts to 10 Volts, the transducer transmits 0 - 2 Volts for 0 - 175 PSI and a 2x amplifier is on the transmitter.



At 0 PSI the transducer transmits 0 Volts. This is amplified to 0 Volts. At half scale, it is read as 2048. At 175 PSI, full scale, the transducer transmits 2 Volts, which is amplified to 4 Volts. The analog input card sees 4 Volts as 70 percent of range or 2867 counts. The engineering units full scale and low scale are determined as follows:

eng units full scale = 43.75 × 10
eng units low scale = 43.75 × (-10)

(175 / 4 = 43.75) The record's field entries to convert this pressure will be as follows:

LINR Linear
EGUF 437.5
EGUL -437.5
EGU PSI

The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = -437.5 + \frac{\text{measured A/D counts}}{4095} * (437.5 - (-437.5))$$

Notice that at low scale the transducer will generate 0 Volts to represent 0 PSI. Because this is half of what the input card accepts, it is input as 2048. Let's plug in the numbers to see the result:

$$-437.5 + (2048/4095) * (437.5 - (-437.5)) = 0$$

Notice that at full scale the transducer will generate 2 volts which represents 175 PSI. The amplifier will change the 2 Volts to 4 Volts. 4 Volts is 14/20 or 70 percent of the I/O card's scale. The input from the I/O card is therefore 2866 (i.e., 0.7 * 4095). Let's plug in the numbers to see the result:

$$-437.5 + (2866/4095) * (437.5 - (-437.5)) = 175PSI$$

We had to adjust the engineering units full scale to adjust for the difference between the transducer with the amplifier affects and the range of the I/O card. We also adjusted the low scale to compensate for the difference between the unipolar transmitter/amplifier and the bipolar analog input card.

Breakpoint Conversions

Now let us consider a non-linear conversion. These are conversions that could be entered as polynomials. As these are more time consuming to execute, a breakpoint table is created that breaks the non-linear conversion into linear segments that are accurate enough.

Breakpoint Table

The breakpoint table is then used to do a piecewise linear conversion. Each piecewise segment of the breakpoint table contains:

Raw Value Start for this segment, Engineering Units at the start.

```
breaktable(typeJdegC) {
   0.000000 0.000000
   365.023224 67.000000
   1000.046448 178.000000
   3007.255859 524.000000
   3543.383789 613.000000
   4042.988281 692.000000
   4101.488281 701.000000
}
```

Breakpoint Conversion Example

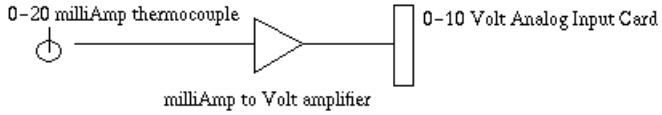
When a new raw value is read, the conversion routine starts from the previously used line segment, compares the raw value start, and either going forward or backward in the table searches the proper segment for this new raw value. Once the proper segment is found, the new engineering units value is the engineering units value at the start of this segment plus the slope of this segment times the position on this segment.

value = eng.units at segment start + (raw value - raw at segment start) * slope

A table that has an entry for each possible raw count is effectively a look up table.

Breakpoint tables are loaded to the IOC using the *dbLoadDatabase* shell function. The slope corresponding to each segment is calculated when the table is loaded. For raw values that exceed the last point in the breakpoint table, the slope of the last segment is used.

In this example the transducer is a thermocouple which transmits 0-20 milliAmps. An amplifier is present which amplifies milliAmps to volts. The I/O card uses a 0-10 Volt interface and a 12-bit ADC. Raw value range would thus be 0 to 4095.



The transducer is transmitting temperature. The database entries in the analog input record that are needed to convert this temperature will be as follows:

LINR typeJdegC
EGUF 0
EGUL 0
EGU DGC

For analog records that use breakpoint tables, the EGUF and EGUL fields are not used in the conversion, so they do not have to be given values.

With this example setup and assuming we get an ADC raw reading of 3500, the formula above would give:

Value = 524.0 + (3500 - 3007) * 0.166 = 605.838 DGC

EPICS Base distribution currently includes lookup tables for J and K thermocouples in degrees F and degrees C.

Other potential applications for a lookup table are e.g. other types of thermocouples, logarithmic output controllers, and exponential transducers. The piece-wise linearization of the signals provides a mechanism for conversion that minimizes the amount of floating point arithmetic required to convert non-linear signals. Additional breakpoint tables can be added to the predefined ones.

Creating Breakpoint Tables

There are two ways to create a new breakpoint table:

1) Simply type in the data for each segment, giving the raw and corresponding engineering unit value for each point in the following format.

```
breaktable(<tablename>) {
  <first point> <first eng units>
  <next point> <next eng units>
  <etc.> <...>
}
```

where the <tablename> is the name of the table, such as typeKdegC, and <first point> is the raw value of the beginning point for each line segment, and <first eng units> is the corresponding engineering unit value. The slope is calculated by the software and should not be specified.

2) Create a file consisting of a table of an arbitrary number of values in engineering units and use the utility called **makeBpt** to convert the table into a breakpoint table. As an example, the contents data file to create the typeJdegC breakpoint table look like this:

```
!header
"typeJdegC" 0 0 700 4095 .5 -210 760 1
!data
-8.096 -8.076 -8.057 <many more numbers>
```

The file name must have the extension .data. The file must first have a header specifying these nine things:

1. Name of breakpoint table in quotes: **"typeJdegC"**
2. Engineering units for 1st breakpoint table entry: **0**
3. Raw value for 1st breakpoint table entry: **0**
4. Highest value desired in engineering units: **700**
5. Raw value corresponding to high value in engineering units: **4095**
6. Allowed error in engineering units: **.5**
7. Engineering units corresponding to first entry in data table: **-210**
8. Engineering units corresponding to last entry in data table: **760**
9. Change in engineering units between data table entries: **1**

The rest of the file contains lines of equally spaced engineering values, with each line no more than 160 characters before the new-line character. The header and the actual table should be specified by **!header** and **!data**, respectively. The file for this data table is called typeJdegC.data, and can be converted to a breakpoint table with the **makeBpt** utility as follows:

unix% makeBpt typeJdegC.data

Alarm Specification

There are two elements to an alarm condition: the alarm *status* and the *severity* of that alarm. Each database record contains its current alarm status and the corresponding severity for that status. The scan task, which detects these alarms, is also capable of generating a message for each change of alarm state. The types of alarms available fall into these categories: scan alarms, read/write alarms, limit alarms, and state alarms. Some of these alarms are configured by the user, and some are automatic which means that they are called by the record support routines on certain conditions, and cannot be changed or configured by the user.

Alarm Severity

An alarm *severity* is used to give weight to the current alarm status. There are four severities:

- NO_ALARM
- MINOR
- MAJOR
- INVALID

NO_ALARM means no alarm has been triggered. An alarm state that needs attention but is not dangerous is a MINOR alarm. In this instance the alarm state is meant to give a warning to the operator. A serious state is a MAJOR alarm. In this instance the operator should give immediate attention to the situation and take corrective action. An INVALID alarm means there's a problem with the data, which can be any one of several problems; for instance, a bad address specification, device communication failure, or signal is over range. In these cases, an alarm severity of INVALID is set. An INVALID alarm can point to a simple configuration problem or a serious operational problem.

For limit alarms and state alarms, the severity can be configured by the user to be MAJOR or MINOR for the a specified state. For instance, an analog record can be configured to trigger a MAJOR alarm when its value exceeds 175.0. In addition to the MAJOR and MINOR severity, the user can choose the NO_ALARM severity, in which case no alarm is generated for that state.

For the other alarm types (i.e., scan, read/write), the severity is always INVALID and not configurable by the user.

Alarm Status

Alarm status is a field common to all records. The field is defined as an enumerated field. The possible states are listed below.

- NO_ALARM: This record is not in alarm
- READ: An INPUT link failed in the device support
- WRITE: An OUTPUT link failed in the device support
- HIHI: An analog value limit alarm
- HIGH: An analog value limit alarm
- LOLO: An analog value limit alarm
- LOW: An analog value limit alarm
- STATE: An digital value state alarm
- COS: An digital value change of state alarm
- COMM: A device support alarm that indicates the device is not communicating
- TIMEOUT: A device sup alarm that indicates the asynchronous device timed out

- HWLIMIT: A device sup alarm that indicates a hardware limit alarm
- CALC: A record support alarm for calculation records indicating a bad calculation
- SCAN: An invalid SCAN field is entered
- LINK: Soft device support for a link failed:no record, bad field, invalid conversion, INVALID alarm severity on the referenced record.
- SOFT
- BAD_SUB
- UDF
- DISABLE
- SIMM
- READ_ACCESS
- WRITE_ACCESS

There are a number of issues with this field and menu.

- The maximum enumerated strings passed through channel access is 16 so nothing past SOFT is seen if the value is not requested by Channel Access as a string.
- Only one state can be true at a time so that the root cause of a problem or multiple problems are masked. This is particularly obvious in the interface between the record support and the device support. The hardware could have some combination of problems and there is no way to see this through the interface provided.
- The list is not complete.
- In short, the ability to see failures through the STAT field are limited. Most problems in the hardware, configuration, or communication are reduced to READ or WRITE error and have their severity set to INVALID. When you have an INVALID alarm severity, some investigation is currently needed to determine the fault. Most EPICS drivers provide a report routine that dumps a large set of diagnostic information. This is a good place to start in these cases.

Alarm Conditions Configured in the Database

When you have a valid value, there are fields in the record that allow the user to configure off normal conditions. For analog values these are limit alarms. For discrete values, these are state alarms.

Limit Alarms

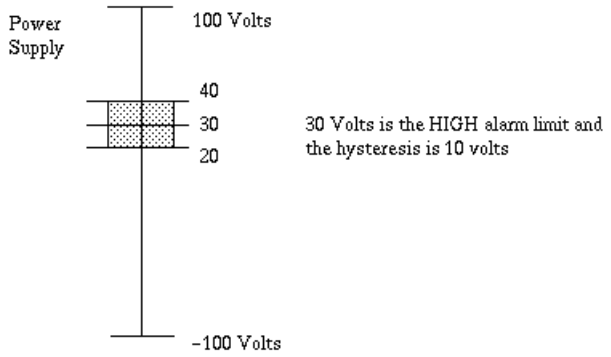
For analog records (this includes such records as the stepper motor record), there are configurable alarm limits. There are two limits for above normal operating range and two limits for the below-limit operating range. Each of these limits has an associated alarm severity, which is configured in the database. If the record's value drops below the low limit and an alarm severity of MAJOR was specified for that limit, then a MAJOR alarm is triggered. When the severity of a limit is set to NO_ALARM, none will be generated, even if the limit entered has been violated.

There are two limits at each end, two low values and two high values, so that a warning can be set off before the value goes into a dangerous condition.

Analog records also contain a hysteresis field, which is also used when determining limit violations. The hysteresis field is the deadband around the alarm limits. The deadband keeps a signal that is hovering at the limit from generating too many alarms. Let's take an example (*Figure 8*) where the range is -100 to 100 volts, the high alarm limit is 30 Volts, and the hysteresis is 10 Volts. If the value is normal and approaches the HIGH alarm limit, an alarm is generated when

the value reaches 30 Volts. This will only go to normal if the value drops below the limit by more than the hysteresis. For instance, if the value changes from 30 to 28 this record will remain in HIGH alarm. Only when the value drops to 20 will this record return to normal state.

Figure 8



State Alarms

For discrete values there are configurable state alarms. In this case a user may configure a certain state to be an alarm condition. Let's consider a cooling fan whose discrete states are high, low, and off. The off state can be configured to be an alarm condition so that whenever the fan is off the record is in a STATE alarm. The severity of this error is configured for each state. In this example, the low state could be a STATE alarm of MINOR severity, and the off state a STATE alarm of MAJOR severity.

Discrete records also have a field in which the user can specify the severity of an unknown state to NO_ALARM, MINOR or MAJOR. Thus, the unknown state alarm is not automatic.

Discrete records also have a field, which can specify an alarm when the record's state changes. Thus, an operator can know when the record's alarm state has changed. If this field specifies NO_ALARM, then a change of state will not trigger a change of state alarm. However, if it specifies either MINOR or MAJOR, a change of state will trigger an alarm with the corresponding severity.

Alarm Handling

A record handles alarms with the NSEV, NSTA, SEVR, and STAT fields. When a software component wants to raise an alarm, it first checks the new alarm state fields: NSTA, new alarm state, and NSEV, new alarm severity. If the severity in the NSEV field is higher than the severity in the current severity field (SEVR), then the software component sets the NSTA and NSEV fields to the severity and alarm state that corresponds to the outstanding alarm. When the record process routine next processes the record, it sets the current alarm state (STAT) and current severity

(SEVR) to the values in the NSEV and NSTA fields. This method of handling alarms ensures that the current severity (STAT) reflects the

highest severity of outstanding alarm conditions instead of simply the last raised alarm. This also means that the if multiple alarms of equal severity are present, the alarm status indicates the first one detected.

In addition, the get_alarm_double() routine can be called to format an alarm message and send it to an alarm handler. The alarm conditions may be monitored by the operator interface by explicitly monitoring the STAT and SEVR fields. All values monitored by the operator interface are returned from the database access with current status information.

Monitor Specification

EPICS provides the methods for clients to subscribe to be informed of changes in a PV; in EPICS vocabulary this method is called "monitor".

In Channel Access, as well as PVAccess clients connect to PVs to put, get, or monitor. There are fields in the EPICS records that help limit the monitors posted to these clients through the CA or PVA Server. These fields most typically apply when the client is monitoring the VAL field of a record. Most other fields post a monitor whenever they are changed. For instance, a put to an alarm limit, causes a monitor to be posted to any client that is monitoring that field. The client can select...

For more information about using monitors, see the Channel Access Reference Guide.

Rate Limits

The inherent rate limit is the rate at which the record is scanned. Monitors are only posted when the record is processed as a minimum. There are currently no mechanisms for the client to rate limit a monitor. If a record is being processed at a much higher rate than an application wants, either the database developer can make a second record at a lower rate and have the client connect to that version of the record or the client can disregard the monitors until the time stamp reflects the change.

Channel Access Deadband Selection

The Channel Access client can set a mask to indicate which alarm change it wants to monitor. There are three: value change, archive change, and alarm change.

Value Change Monitors

The value change monitors are typically sent whenever a field in the database changes. The VAL field is the exception. If the MDEL field is set, then the VAL field is sent when a monitor is set, and then only sent again, when the VAL field has changed by MDEL.

..note:

```
A MDEL of 0 sends a monitor whenever the VAL fields changes and an MDEL of -1
sends a monitor whenever the record is processed as the MDEL is applied
to the absolute value of the difference between the previous scan and
the current scan. An MDEL of -1 is useful for scalars that are triggered
and a positive indication that the trigger occurred is required.
```

Archive Change Monitors

The archive change monitors are typically sent whenever a field in the database changes. The VAL field is the exception. If the ADEL field is set, then the VAL field is sent when a monitor is set, and then only sent again, when the VAL field has changed by ADEL.

Alarm Change Monitors

The alarm change monitors are only sent when the alarm severity or status change. As there are filters on the alarm condition checking, the change of alarm status or severity is already filtered through those mechanisms. These are described in *Alarm Specification*.

Metadata Changes

When a Channel Access Client connects to a field, it typically requests some metadata related to that field. One case is a connection from an operator interface typically requests metadata that includes: display limits, control limits, and display information such as precision and engineering units. If any of the fields in a record that are included in this metadata change after the connection is made, the client is not informed and therefore this is not reflected unless the client disconnects and reconnects. A new flag is being added to the Channel Access Client to support posting a monitor to the client whenever any of this metadata changes. Clients can then request the metadata and reflect the change.

Stay tuned for this improvement in the record support and channel access clients.

Client specific Filtering

Several situation have come up that would be useful. These include event filtering, rate guarantee, rate limit, and value change.

Event Filtering

There are several cases where a monitor was sent from a channel only when a specific event was true. For instance, there are diagnostics that are read at 1 kHz. A control program may only want this information when the machine is producing a particular beam such as a linac that has several injectors and beam lines. These are virtual machines that want to be notified when the machine is in their mode. These modes can be interleaved at 60 Hz in some cases. A fault analysis tool may only be interested in all of this data when a fault occurs and the beam is dumped.

There are two efforts here: one at LANL and one from ANL/BNL. These should be discussed in the near future.

Rate Guarantee

Some clients may want to receive a monitor at a given rate. Binary inputs that only notify on change of state may not post a monitor for a very long time. Some clients may prefer to have a notification at some rate even when the value is not changing.

Rate Limit

There is a limit to the rate that most clients care to be notified. Currently, only the SCAN period limits this. A user-imposed limit is needed in some cases such as a data archiver that would only want this channel at 1 Hz (all channels on the same 1 msec in this case).

Value Change

Different clients may have a need to set different deadbands among them. No specific case is cited.

Control Specification

A control loop is a set of database records used to maintain control autonomously. Each output record has two fields that are help implement this independent control: the desired output location field (DOL) and the output mode select field (OMSL). The OMSL field has two mode choices: closed_loop or supervisory. When the closed loop mode is chosen, the desired output is retrieved from the location specified by the DOL field and placed into the VAL field. When the supervisory mode is chosen, the desired output value is the VAL field. In supervisory mode the DOL link is not retrieved. In the supervisory mode, VAL is set typically by the operator through a Channel Access "Put".

Closing an Analog Control Loop

In a simple control loop an analog input record reads the value of a process variable or PV. The operator sets the Setpoint in the PID record. Then, a PID record retrieves the value from the analog input record and computes the error - the difference between the readback and the setpoint. The PID record computes the new output setting to move the process variable toward the setpoint. The analog output record gets the value from the PID through the DOL when the OMSL is closed_loop. It sets the new output and on the next period repeats this process.

Configuring an Interlock

When certain conditions become true in the process, it may trip an interlock. The result of this interlock is to move something into a safe state or to mitigate damage by taking some action. One example is the closing of a vacuum valve to isolate a vacuum loss. When a vacuum reading in one region of a machine is not at the operating range, an interlock is used to either close a valve and prohibit it from being open. This can be implemented by reading several vacuum gauges in an area into a calculation record. The expression in the calculation record can express the condition that permits the valve to open. The result of the expression is then referenced to the DOL field of a binary output record that controls the valve. If the binary output has the OMSL field set to closed_loop it sets the valve to the value of the calculation record. If it is set to supervisory, the operator can override the interlock and control the valve directly.

1.13.4 Database Definition

Tags: developer advanced

Overview

This chapter describes database definitions. The following definitions are described:

- Menu
- Record Type
- Device
- Driver
- Registrar
- Variable
- Function

- Breakpoint Table
- Record Instance

Record Instances are fundamentally different from the other definitions. A file containing record instances should never contain any of the other definitions and vice-versa. Thus the following convention is followed:

Database Definition File
A file that contains any type of definition except record instances.

Record Instance File
A file that contains only record instance definitions.

This chapter also describes utility programs which operate on these definitions.

Any combination of definitions can appear in a single file or in a set of files related to each other via include statements.

Summary of Database Syntax

The following summarizes the Database Definition syntax:

```
path "path"
addpath "path"
include "filename"
#comment
menu(name) {
    include "filename"
    choice(choice_name, "choice_value")
    ...
}

recordtype(record_type) {}

recordtype(record_type) {
    include "filename"
    field(field_name, field_type) {
        asl(asl_level)
        initial("init_value")
        promptgroup("group_name")
        prompt("prompt_value")
        special(special_value)
        pp(pp_value)
        interest(interest_level)
        base(base_type)
        size(size_value)
        extra("extra_info")
        menu(name)
        prop(yesno)
    }
    %C_declaration
    ...
}

device(record_type, link_type, dset_name, "choice_string")

driver(drvet_name)
```

(continues on next page)

(continued from previous page)

```
registrar(function_name)

variable(variable_name)

breaktable(name) {
    raw_value eng_value
    ...
}
```

The Following defines a Record Instance

```
record(record_type, record_name) {
    include "filename"
    field(field_name, "value")
    alias(alias_name)
    info(info_name, "value")
    ...
}
alias(record_name,alias_name)
```

General Rules for Database Definition

Keywords

The following are keywords, i.e. they may not be used as values unless they are enclosed in quotes:

```
path
addpath
include
menu
choice
recordtype
field
device
driver
registrar
function
variable
breaktable
record
grecord
info
alias
```

Unquoted Strings

In the summary section, some values are shown as quoted strings and some unquoted. The actual rule is that any string consisting of only the following characters does not need to be quoted unless it contains one of the above keywords:

```
a-z A-Z 0-9 _ + - : . [ ] < > ;
```

These are all legal characters for process variable names, although `.` is not allowed in a record name since it separates the record from the field name in a PV name. Thus in many cases quotes are not needed around record or field names in database files. Any string containing a macro does need to be quoted though.

Quoted Strings

A quoted string can contain any ascii character except the quote character ". The quote character itself can given by using a back-slash (\) as an escape character. For example "\"" is a quoted string containing a single double-quote character.

Macro Substitution

Macro substitutions are permitted inside quoted strings. Macro instances take the form:

```
$(name)
```

or

```
${name}
```

There is no distinction between the use of parentheses or braces for delimiters, although the opening and closing characters must match for each macro instance. A macro name can be constructed using other macros, for example:

```
$(name_$(sel))
```

A macro instance can also provide a default value that is used when no macro with the given name has been defined. The default value can itself be defined in terms of other macros if desired, but may not contain any unescaped comma characters. The syntax for specifying a default value is as follows:

```
$(name=default)
```

Finally macro instances can also set the values of other macros which may (temporarily) override any existing values for those macros, but the new values are in scope only for the duration of the expansion of this particular macro instance. These definitions consist of `name=value` sequences separated by commas, for example:

```
$(abcd=$(a)$(b)$(c)$(d),a=A,b=B,c=C,d=D)
```

Escape Sequences

The database routines translate standard C escape sequences inside database field value strings only. The standard C escape sequences supported are:

```
\a \b \f \n \r \t \v \\ \' \" \ooo \xhh
```

`\ooo` represents an octal number with 1, 2, or 3 digits. `\xhh` represents a hexadecimal number which may have any number of hex digits, although only the last 2 will be represented in the character generated.

Comments

The comment symbol is "#". Whenever the comment symbol appears outside of a quoted string, it and all subsequent characters through the end of the line will be ignored.

Define before referencing

In general items cannot be referenced until they have been defined. For example a `device` definition cannot appear until the `recordtype` that it references has been defined or at least declared. Another example is that a record instance cannot appear until its associated record type has been defined.

One notable exception to this rule is that within a `recordtype` definition a menu field may reference a menu that has not been included directly by the record's `.dbd` file.

Multiple Definitions

If a menu, device, driver, or breakpoint table is defined more than once, then only the first instance will be used. Subsequent definitions may be compared to the first one and an error reported if they are different (the `dbdExpand.pl` program does this, the IOC currently does not). Record type definitions may only be loaded once; duplicates will cause an error even if the later definitions are identical to the first. However a record type declaration may be used in place of the record type definition in `.dbd` files that define device support for that type.

Record instance definitions are (normally) cumulative, so multiple instances of the same record may be loaded and each time a field value is encountered it replaces the previous value.

Filename Extensions

By convention:

- Record instances files have the extension "`.db`" or "`.vdb`" if the file also contains visual layout information
- Database definition files have the extension "`.dbd`"

Database Definition Statements

`path` `addpath` – Path Definition

Format

```
path "dir:dir...:dir"
addpath "dir:dir...:dir"
```

The path string follows the standard convention for the operating system, i.e. directory names are separated by a colon ":" on Unix and a semicolon ";" on Windows.

The `path` statement specifies the current search path for use when loading database and database definition files. The `addpath` statement appends directories to the current path. The path is used to locate the initial database file and included files. An empty path component at the beginning, middle, or end of a non-empty path string means search the current directory. For example:

```
nnn::mmm    # Current directory is between nnn and mmm
:nnn        # Current directory is first
nnn:        # Current directory is last
```

Utilities which load database files (`dbExpand`, `dbLoadDatabase`, etc.) allow the user to specify an initial path. The `path` and `addpath` commands can be used to change or extend that initial path.

The initial path is determined as follows:

1. If path is provided with the command, it is used. Else:
2. If the environment variable `EPICS_DB_INCLUDE_PATH` is defined, it is used. Else:
3. the path is ".", i.e. the current directory.

The search path is not used at all if the filename being searched for contains a / or \ character. The first instance of the specified filename is used.

`include` – Include Statement

Format

```
include "filename"
```

An include statement can appear at any place shown in the summary. It uses the search path as described above to locate the named file.

`menu` – Menu Definition

Format

```
menu(name) {
    choice(choice_name, "choice_string")
    ...
}
```

Definitions

name
Name for menu. This is the unique name identifying the menu. If duplicate definitions are specified, only the first is used.

choice_name
The name used in the `enum` generated by `dbdToMenuH.pl` or `dbdToRecordtypeH.pl`. This must be a legal C/C++ identifier.

choice_string
The text string associated with this particular choice.

Example

```
menu(menuYesNo) {
    choice(menuYesNoNO, "NO")
    choice(menuYesNoYES, "YES")
}
```

`recordtype` – Record Type Definition

Format

```
recordtype(record_type) {}

recordtype(record_type) {
    field(field_name, field_type) {
        asl(as_level)
        initial("init_value")
        promptgroup("group_name")
        prompt("prompt_value")
        special(special_value)
        pp(pp_value)
        interest(interest_level)
        base(base_type)
        size(size_value)
        extra("extra_info")
        menu(name)
        prop(yesno)
    }
    %C_declaration
    ...
}
```

A record type statement that provides no field descriptions is a declaration, analagous to a function declaration (prototype) or forward definition in C. It allows the given record type name to be used in circumstances where the full record type definition is not needed.

Field Descriptor Rules

asl
Sets the Access Security Level for the field. Access Security is discussed in chapter *[Access Security]*.

initial
Provides an initial (default) value for the field.

promptgroup
The group to which the field belongs, for database configuration tools.

prompt
A prompt string for database configuration tools. Optional if `promptgroup` is not defined.

special
If specified, special processing is required for this field at run time.

pp
Whether a passive record should be processed when Channel Access writes to this field.

interest
Interest level for the field.

base
For integer fields, the number base to use when converting the field value to a string.

size
Must be specified for `DBF_STRING` fields.

extra
Must be specified for `DBF_NOACCESS` fields.

menu
Must be specified for `DBF_MENU` fields. It is the name of the associated menu.

prop
Must be `YES` or `NO` (default). Indicates that the field holds Channel Access meta-data.

Definitions

record_type
The unique name of the record type. Duplicate definitions are not allowed and will be rejected.

field_name
The field name, which must be a valid C and C++ identifier. When include files are generated, the field name is converted to lower case for use as the record structure member name. If the lower-case version of the field name is a C or C++ keyword, the original name will be used for the structure member name instead. Previous versions of EPICS required the field name be a maximum of four all upper-case characters, but these restrictions no longer apply.

field_type
This must be one of the following values:

- `DBF_STRING`
- `DBF_CHAR`, `DBF_UCHAR`
- `DBF_SHORT`, `DBF_USHORT`
- `DBF_LONG`, `DBF_ULONG`
- `DBF_FLOAT`, `DBF_DOUBLE`

- DBF_ENUM, DBF_MENU, DBF_DEVICE
- DBF_INLINK, DBF_OUTLINK, DBF_FWDLINK
- DBF_NOACCESS

as_level
This must be one of the following values:

- ASL0
- ASL1 (default value)

Fields which operators normally change are assigned ASL0. Other fields are assigned ASL1. For example, the VAL field of an analog output record is assigned ASL0 and all other fields ASL1. This is because only the VAL field should be modified during normal operations.

init_value
A legal value for data type.

prompt_value
A prompt value for database configuration tools.

group_name
A string used by database configuration tools (DCTs) to group related fields together.

A promptgroup should only be set for fields that can sensibly be configured in a record instance file.

The set of group names is no longer fixed. In earlier versions of Base the predefined set of choices beginning GUI_ were the only group names permitted. Now the group name strings found in the database definition file are collected and stored in a global list. The strings given for group names must match exactly for fields to be grouped together.

To support sorting and handling of groups, the names used in Base have the following conventions:

- Names start with a two-digit number followed by a space-dash-space sequence.
- Names are designed to be presented in ascending numerical order.
- The group name (or possibly just the part following the dash) may be displayed by the tool as a title for the group.
- In many-of-the-same-kind cases (e.g. 21 similar inputs) fields are distributed over multiple groups. Once-only fields appear in groups numbered in multiples of 5 or 10. The groups with the multiple instances follow in +1 increments. This allows more sophisticated treatment, e.g. showing the first group open and the other groups collapsed.

Record types may define their own group names. However, to improve consistency, records should use the following names from Base where possible. (This set also demonstrates that the group names used in different record types may share the same number.)

- General fields that are common to all or many record types
- Scanning mechanism, priority and related properties
- Record type specific behavior and processing action
- Links and related properties
- Input links and properties
- Output links and properties
- Conversion between raw and engineering values
- Alarm related properties, severities and thresholds

- Client related configuration, strings, deadbands
- Simulation mode related properties

NOTE: Older versions of Base contained a header file `guigroup.h` defining a fixed set of group names and their matching index numbers. That header file has been removed. The static database access library now provides functions to convert between group index keys and the associated group name strings. See *[subsec:Get Field Prompt]* for details.

special_value
Must be one of the following:

- `SPC_MOD` – Notify record support when modified. The record support `special` routine will be called whenever the field is modified by the database access routines.
- `SPC_NOMOD` – No external modifications allowed. This value disables external writes to the field, so it can only be set by the record or device support module.
- `SPC_DBADDR` – Use this if the record support's `cvt_dbaddr` routine should be called to adjust the field description when code outside of the record or device support makes a connection to the field.

The following values are for database common fields. They must *not* be used for record specific fields:

- `SPC_SCAN` – Scan related field.
- `SPC_ALARMACK` – Alarm acknowledgment field.
- `SPC_AS` – Access security field.

The following values are deprecated, use `SPC_MOD` instead:

- An integer value greater than 103.
- `SPC_RESET` – a reset field is being modified.
- `SPC_LINCONV` – A linear conversion field is being modified.
- `SPC_CALC` – A calc field is being modified.

pp_value
Should a passive record be processed when Channel Access writes to this field? The allowed values are:

- `FALSE` (default)
- `TRUE`

interest_level
An interest level for the `dbpr` command.

base
For integer type fields, the default base. The legal values are:

- `DECIMAL` (Default)
- `HEX`

size_value
The number of characters for a `DBF_STRING` field.

extra_info
For `DBF_NOACCESS` fields, this is the C language definition for the field. The definition must end with the field-name in lower case.

%C_declaration
A percent sign `%` inside the record body introduces a line of code that is to be included in the generated C header file.

Example

The following is the definition of the event record type:

```
recordtype(event) {
    include "dbCommon.dbd"
    field(VAL,DBF_STRING) {
        prompt("Event Name To Post")
        promptgroup("40 - Input")
        special(SPC_MOD)
        asl(ASL0)
        size(40)
    }
    field(EPVT, DBF_NOACCESS) {
        prompt("Event private")
        special(SPC_NOMOD)
        interest(4)
        extra("EVENTPVT epvt")
    }
    field(INP,DBF_INLINK) {
        prompt("Input Specification")
        promptgroup("40 - Input")
        interest(1)
    }
    field(SIOL,DBF_INLINK) {
        prompt("Sim Input Specifctn")
        promptgroup("90 - Simulate")
        interest(1)
    }
    field(SVAL,DBF_STRING) {
        prompt("Simulation Value")
        size(40)
    }
    field(SIML,DBF_INLINK) {
        prompt("Sim Mode Location")
        promptgroup("90 - Simulate")
        interest(1)
    }
    field(SIMM,DBF_MENU) {
        prompt("Simulation Mode")
        interest(1)
        menu(menuYesNo)
    }
    field(SIMS,DBF_MENU) {
        prompt("Sim mode Alarm Svrty")
        promptgroup("90 - Simulate")
        interest(2)
        menu(menuAlarmSevr)
    }
}
```

`device` – Device Support Declaration

Format

```
device(record_type, link_type, dset_name, "choice_string")
```

Definitions

record_type
Record type. The combination of `record_type` and `choice_string` must be unique. If the same combination appears more than once, only the first definition is used.

link_type
Link type. This must be one of the following:

- `CONSTANT`
- `PV_LINK`
- `VME_IO`
- `CAMAC_IO`
- `AB_IO`
- `GPIB_IO`
- `BITBUS_IO`
- `INST_IO`
- `BBGPIB_IO`
- `RF_IO`
- `VXI_IO`

dset_name
The name of the device support entry table for this device support.

choice_string
The `DTYP` choice string for this device support. A `choice_string` value may be reused for different record types, but must be unique for each specific record type.

Examples

```
device(ai,CONSTANT,devAiSoft,"Soft Channel")
device(ai,VME_IO,devAiXy566Se,"XYCOM-566 SE Scanned")
```

`driver` – Driver Declaration

Format

```
driver(drvet_name)
```

Definitions

drvet_name
If duplicates are defined, only the first is used.

Examples

```
driver(drvVxi)
driver(drvXy210)
```

`registrar` – Registrar Declaration

Format

```
registrar(function_name)
```

Definitions

function_name
The name of an C function that accepts no arguments, returns `void` and has been marked in its source file with an `epicsExportRegistrar` declaration, e.g.

```
static void myRegistrar(void);
epicsExportRegistrar(myRegistrar);
```

This can be used to register functions for use by subroutine records or that can be invoked from iocsh. The example application described in Section *[Example IOC Application]*, "Example IOC Application" gives an example of how to register functions for subroutine records.

Example

```
registrar(myRegistrar)
```

`variable` – Variable Declaration

Format

```
variable(variable_name[, type])
```

Definitions

variable_name
The name of a C variable which has been marked in its source file with an `epicsExportAddress` declaration.

type
The C variable's type. If not present, `int` is assumed. Currently only `int` and `double` variables are supported.

This registers a diagnostic/configuration variable for device or driver support or a subroutine record subroutine. This variable can be read and set with the iocsh `var` command (see Section *[Utility Commands]*. The example application described in Section *[Example IOC Application]* shows how to register a debug variable for use in a subroutine record.

Example

In an application C source file:

```
#include <epicsExport.h>

static double myParameter;
epicsExportAddress(double, myParameter);
```

In an application database definition file:

```
variable(myParameter, double)
```

`function` – Function Declaration

Format

```
function(function_name)
```

Definitions

function_name
The name of a C function which has been exported from its source file with an `epicsRegisterFunction` declaration.

This registers a function so that it can be found in the function registry for use by record types such as sub or aSub which refer to the function by name. The example application described in Section *[Example IOC Application]* shows how to register functions for a subroutine record.

Example

In an application C source file:

```
#include <registryFunction.h>
#include <epicsExport.h>

static long myFunction(void *argp) {
    /* my code ... */
}
epicsRegisterFunction(myFunction);
```

In an application database definition file:

```
function(myFunction)
```

breaktable – Breakpoint Table

Format

```
breaktable(name) {
    raw_value eng_value
    ...
}
```

Definitions

name
: Name, which must be alpha-numeric, of the breakpoint table. If duplicates are specified the first is used.

raw_value
: The raw value, i.e. the actual ADC value associated with the beginning of the interval.

eng_value
: The engineering value associated with the beginning of the interval.

Example

```
breaktable(typeJdegC) {
    0.000000 0.000000
    365.023224 67.000000
    1000.046448 178.000000
    3007.255859 524.000000
    3543.383789 613.000000
    4042.988281 692.000000
    4101.488281 701.000000
}
```

record – Record Instance

Format

```
record(record_type, record_name) {
    alias(alias_name)
    field(field_name, "field_value")
    info(info_name, "info_value")
    ...
}
alias(record_name, alias_name)
```

Definitions

record_type
The record type, or "*" (see discussion under record_name below).

record_name
The record name. This must be composed out of only the following characters:

```
a-z A-Z 0-9 _ - + : [ ] < > ;
```

NOTE: If macro substitutions are used the name must be quoted.

Duplicate definitions are normally allowed for a record as long as the record type is the same. The last value given for each field is the value used. If the duplicate definitions are being used and the record has already been loaded, subsequent definitions may use "*" in place of the record type in the record instance.

The variable dbRecordsOnceOnly can be set to any non-zero value using the iocsh var command to make loading duplicate record definitions into the IOC illegal.

alias_name
An alternate name for the record, following the same rules as the record name.

field_name
A field name.

field_value
A value for the named field, appropriate for its particular field type. When given inside double quotes the field value string may contain escaped characters which will be translated appropriately when loading the database. See section *1.3.5* for the list of escaped characters supported. Permitted values for the various field types are as follows:

- DBF_STRING
  Any ASCII string. If it exceeds the field length, it will be truncated.
- DBF_CHAR, DBF_UCHAR, DBF_SHORT, DBF_USHORT, DBF_LONG, DBF_ULONG
  A string that represents a valid integer. The standard C conventions are applied, i.e. a leading 0 means the value is given in octal and a leading 0x means that value is given in hex.
- DBF_FLOAT, DBF_DOUBLE
  The string must represent a valid floating point number. Infinities or NaN are also allowed.
- DBF_MENU
  The string must be one of the valid choices for the associated menu.
- DBF_DEVICE

The string must be one of the valid device choice strings.

- DBF_INLINK, DBF_OUTLINK, DBF_FWDLINK
  NOTES:
  - If the field name is INP or OUT then this field is associated with DTYP, and the permitted values are determined by the link type of the device support selected by the current DTYP choice string. Other DBF_INLINK and DBF_OUTLINK fields must be either CONSTANT or PV_LINKs.
  - A device support that specifies a link type of CONSTANT can be given either a constant or a PV_LINK.

  The allowed values for the field depend on the device support's link type as follows:
  - CONSTANT
    A numeric literal, valid for the field type it is to be read into.
  - PV_LINK
    A value of the form:

    ```
    record.field process maximize
    ```

    record is the name of a record that exists in this or another IOC.

    The .field, process, and maximize parts are all optional.

    The default value for .field is .VAL.

    process can have one of the following values:

    * NPP – No Process Passive (Default)
    * PP – Process Passive
    * CA – Force link to be a channel access link
    * CP – CA and process on monitor
    * CPP – CA and process on monitor if record is passive

      NOTES:

      CP and CPP are valid only for DBF_INLINK fields.

      DBF_FWDLINK fields can use PP or CA. If a DBF_FWDLINK is a channel access link it must reference the target record's PROC field.

    maximize can have one of the following values:

    * NMS – No Maximize Severity (Default)
    * MS – Maximize Severity
    * MSS – Maximize Severity and Status
    * MSI – Maximize Severity if Invalid
  - VME_IO
    #Ccard Ssignal @parm

    card – the card number of associated hardware module
    signal – signal on card
    parm – An arbitrary character string of up to 31 characters. This field is optional and is device specific.

- CAMAC_IO
  #Bbranch Ccrate Nstation Asubaddress Ffunction @parm

  branch, crate, station, subaddress, and function should be obvious to camac users. subaddress and function are optional (0 if not given). parm is also optional and is device specific (25 characters max).

- AB_IO
  #Llink Aadapter Ccard Ssignal @parm

  link – Scanner, i.e. vme scanner number
  adapter – Adapter. Allen Bradley also calls this rack
  card – Card within Allen Bradley Chassis
  signal – signal on card
  parm – optional device-specific character string (27 char max)

- GPIB_IO
  #Llink Aaddr @parm

  link – gpib link, i.e. interface
  addr – GPIB address
  parm – device-specific character string (31 char max)

- BITBUS_IO
  #Llink Nnode Pport Ssignal @parm

  link – link, i.e. vme bitbus interface
  node – bitbus node
  port – port on the node
  signal – signal on port
  parm – device specific-character string (31 char max)

- INST_IO @parm

  parm – Device dependent character string

- BBGPIB_IO
  #Llink Bbbaddr Ggpibaddr @parm

  link – link, i.e. vme bitbus interface
  bbadddr – bitbus address
  gpibaddr – gpib address
  parm – optional device-specific character string (31 char max)

- RF_IO

#Rcryo Mmicro Ddataset Eelement

– VXI_IO
#Vframe Cslot Ssignal @parm (Dynamic addressing)
or
#Vla Signal @parm (Static Addressing)

frame – VXI frame number
slot – Slot within VXI frame
la – Logical Address
signal – Signal Number
parm – device specific character string(25 char max)

info_name
The name of an Information Item related to this record. See section *1.5* below for more on Information Items.

info_value
Any ASCII string. IOC applications using this information item may place additional restrictions on the contents of the string.

Examples

```
record(ai,STS_AbAiMaS0) {
    field(SCAN,".1 second")
    field(DTYP,"AB-1771IFE-4to20MA")
    field(INP,"#L0 A2 C0 S0 F0 @")
    field(PREC,"4")
    field(LINR,"LINEAR")
    field(EGUF,"20")
    field(EGUL,"4")
    field(EGU,"MilliAmps")
    field(HOPR,"20")
    field(LOPR,"4")
}
record(ao,STS_AbAoMaC1S0) {
    field(DTYP,"AB-1771OFE")
    field(OUT,"#L0 A2 C1 S0 F0 @")
    field(LINR,"LINEAR")
    field(EGUF,"20")
    field(EGUL,"4")
    field(EGU,"MilliAmp")
    field(DRVH,"20")
    field(DRVL,"4")
    field(HOPR,"20")
    field(LOPR,"4")
    info(autosaveFields,"VAL")
}
record(bi,STS_AbDiA0C0S0) {
    field(SCAN,"I/O Intr")
    field(DTYP,"AB-Binary Input")
```

(continues on next page)

(continued from previous page)

```
    field(INP,"#L0 A0 C0 S0 F0 @")
    field(ZNAM,"Off")
    field(ONAM,"On")
}
```

Record Information Item

Information items provide a way to attach named string values to individual record instances that are loaded at the same time as the record definition. They can be attached to any record without having to modify the record type, and can be retrieved by programs running on the IOC (they are not visible via Channel Access at all). Each item attached to a single record must have a unique name by which it is addressed, and database access provides routines to allow a record's info items to be scanned, searched for, retrieved and set. At runtime a `void*` pointer can also be associated with each item, although only the string value can be initialized from the record definition when the database is loaded.

Record Attributes

Each record type can have any number of record attributes. Each attribute is a psuedo field that can be accessed via database and channel access. Each attribute has a name that acts like a field name but returns the same value for all instances of the record type. Two attributes are generated automatically for each record type: `RTYP` and `VERS`. The value for `RTYP` is the record type name. The default value for `VERS` is "none specified", which can be changed by record support. Record support can call the following routine to create new attributes or change existing attributes:

```
long dbPutAttribute(char *rtype, char *name, char *value);
```

The arguments are:

`rtype` – The name of recordtype.

`name` – The attribute name, i.e. the psuedo field name.

`value` – The value assigned to the attribute.

Breakpoint Tables – Discussion

The menu `menuConvert` is used for field `LINR` of the `ai` and `ao` records. These records allow raw data to be converted to/from engineering units via one of the following:

1. No Conversion.
2. Slope Conversion.
3. Linear Conversion.
4. Breakpoint table.

Other record types can also use this feature. The first choice specifies no conversion; the second and third are both linear conversions, the difference being that for Slope conversion the user specifies the conversion slope and offset values directly, whereas for Linear conversions these are calculated by the device support from the requested Engineering Units range and the device support's knowledge of the hardware conversion range. The remaining choices are assumed to be the names of breakpoint tables. If a breakpoint table is chosen, the record support modules calls `cvtRawToEngBpt` or `cvtEngToRawBpt`. You can look at the `ai` and `ao` record support modules for details.

If a user wants to add additional breakpoint tables, then the following should be done:

- Copy the `menuConvert.dbd` file from EPICS `base/src/ioc/bpt`

- Add definitions for new breakpoint tables to the end
- Make sure modified `menuConvert.dbd` is loaded into the IOC instead of EPICS version.

It is only necessary to load a breakpoint file if a record instance actually chooses it. It should also be mentioned that the Allen Bradley IXE device support misuses the `LINR` field. If you use this module, it is very important that you do not change any of the EPICS supplied definitions in `menuConvert.dbd`. Just add your definitions at the end.

If a breakpoint table is chosen, then the corresponding breakpoint file must be loaded into the IOC before `iocInit` is called.

Normally, it is desirable to directly create the breakpoint tables. However, sometimes it is desirable to create a breakpoint table from a table of raw values representing equally spaced engineering units. A good example is the Thermocouple tables in the OMEGA Engineering, INC Temperature Measurement Handbook. A tool `makeBpt` is provided to convert such data to a breakpoint table.

The format for generating a breakpoint table from a data table of raw values corresponding to equally spaced engineering values is:

```
!comment line
<header line>
<data table>
```

The header line contains the following information:

Name
An alphanumeric ascii string specifying the breakpoint table name

Low Value Eng
Engineering Units Value for first breakpoint table entry

Low Value Raw
Raw value for first breakpoint table entry

High Value Eng
Engineering Units: Highest Value desired

High Value Raw
Raw Value for High Value Eng

Error
Allowed error (Engineering Units)

First Table
Engineering units corresponding to first data table entry

Last Table
Engineering units corresponding to last data table entry

Delta Table
Change in engineering units per data table entry

An example definition is:

```
"TypeKdegF" 32 0 1832 4095 1.0 -454 2500 1
<data table>
```

The breakpoint table can be generated by executing

```
makeBpt bptXXX.data
```

The input file must have the extension of data. The output filename is the same as the input filename with the extension of .dbd.

Another way to create the breakpoint table is to include the following definition in a Makefile:

```
BPTS += bptXXX.dbd
```

NOTE: This requires the naming convention that all data tables are of the form bpt<name>.data and a breakpoint table bpt<name>.dbd.

Menu and Record Type Include File Generation.

Introduction

Given a file containing menu definitions, the program dbdToMenuH.pl generates a C/C++ header file for use by code which needs those menus. Given a file containing any combination of menu definitions and record type definitions, the program dbdToRecordtypeH.pl generates a C/C++ header file for use by any code which needs those menus and record type.

EPICS Base uses the following conventions for managing menu and recordtype definitions. Users generating local record types are encouraged to follow these.

- Each menu that is used by fields in database common (for example menuScan) or is of global use (for example menuYesNo) should be defined in its own file. The name of the file is the same as the menu name, with an extension of .dbd. The name of the generated include file is the menu name, with an extension of .h. Thus menuScan is defined in a file menuScan.dbd and the generated include file is named menuScan.h
- Each record type is defined in its own file. This file should also contain any menu definitions that are used only by that record type. Menus that are specific to one particular record type should use that record type name as a prefix to the menu name. The name of the file is the same as the record type, followed by Record.dbd. The name of the generated include file is the same as the .dbd file but with an extension of .h. Thus the record type ao is defined in a file aoRecord.dbd and the generated include file is named aoRecord.h. Since aoRecord has a private menu called aoOIF, the dbd file and the generated include file will have definitions for this menu. Thus for each record type, there are two source files (xxxRecord.dbd and xxxRecord.c) and one generated file (xxxRecord.h).

Note that developers don't normally execute the dbdToMenuH.pl or dbdToRecordtypeH.pl programs manually. If the proper naming conventions are used, it is only necessary to add definitions to the appropriate Makefile. Consult the chapter on the EPICS Build Facility for details.

dbdToMenuH.pl

This tool is executed as follows:

```
dbdToMenuH.pl [-D] [-I dir] [-o menu.h] menu.dbd [menu.h]
```

It reads in the input file menu.dbd and generates a C/C++ header file containing enumerated type definitions for the menus found in the input file.

Multiple -I options can be provided to specify directories that must be searched when looking for included files. If no output filename is specified with the -o menu.h option or as a final command-line parameter, then the output filename will be constructed from the input filename, replacing .dbd with .h.

The -D option causes the program to output Makefile dependency information for the output file to standard output, instead of actually performing the functions describe above.

For example menuPriority.dbd, which contains the definitions for processing priority contains:

```
menu(menuPriority) {
    choice(menuPriorityLOW,"LOW")
    choice(menuPriorityMEDIUM,"MEDIUM")
    choice(menuPriorityHIGH,"HIGH")
}
```

The include file menuPriority.h that is generated contains:

```
/* menuPriority.h generated from menuPriority.dbd */

#ifndef INC_menuPriority_H
#define INC_menuPriority_H

typedef enum {
    menuPriorityLOW                 /* LOW */,
    menuPriorityMEDIUM              /* MEDIUM */,
    menuPriorityHIGH                /* HIGH */,
    menuPriority_NUM_CHOICES
} menuPriority;

#endif /* INC_menuPriority_H */
```

Any code that needs the priority menu values should include this file and make use of these definitions.

dbdToRecordtypeH.pl

This tool is executed as follows:

```
dbdTorecordtypeH.pl [-D] [-I dir] [-o xRecord.h] xRecord.dbd [xRecord.h]
```

It reads in the input file xRecord.dhd and generates a C/C++ header file which defines the in-memory structure of the given record type and provides other associated information for the compiler. If the input file contains any menu definitions, they will also be converted into enumerated type definitions in the output file.

Multiple -I options can be provided to specify directories that must be searched when looking for included files. If no output filename is specified with the -o xRecord.h option or as a final command-line parameter then the output filename will be constructed from the input filename, replacing .dbd with .h.

The -D option causes the program to output Makefile dependency information for the output file to standard output, instead of actually performing the functions describe above.

For example aoRecord.dbd, which contains the definitions for the analog output record contains:

```
menu(aoOIF) {
    choice(aoOIF_Full,"Full")
    choice(aoOIF_Incremental,"Incremental")
}
recordtype(ao) {
    include "dbCommon.dbd"
    field(VAL,DBF_DOUBLE) {
        prompt("Desired Output")
        promptgroup("50 - Output")
```

(continues on next page)

(continued from previous page)

```
        asl(ASL0)
        pp(TRUE)
    }
    field(OVAL,DBF_DOUBLE) {
        prompt("Output Value")
    }
    ... many more field definitions
}
```

The include file `aoRecord.h` that is generated contains:

```
/* aoRecord.h generated from aoRecord.dbd */

#ifndef INC_aoRecord_H
#define INC_aoRecord_H

#include "epicsTypes.h"
#include "link.h"
#include "epicsMutex.h"
#include "ellLib.h"
#include "epicsTime.h"

typedef enum {
    aoOIF_Full                      /* Full */,
    aoOIF_Incremental               /* Incremental */,
    aoOIF_NUM_CHOICES
} aoOIF;

typedef struct aoRecord {
    char                name[61];   /* Record Name */
    ... define remaining fields from database common
    epicsFloat64        val;        /* Desired Output */
    epicsFloat64        oval;       /* Output Value */
    ... define remaining record specific fields
} aoRecord;

typedef enum {
    aoRecordNAME = 0,
    aoRecordDESC = 1,
    ... indices for remaining fields in database common
    aoRecordVAL = 43,
    aoRecordOVAL = 44,
    ... indices for remaining record specific fields
} aoFieldIndex;

#ifdef GEN_SIZE_OFFSET

#ifdef __cplusplus
extern "C" {
#endif
#include <epicsExport.h>
static int aoRecordSizeOffset(dbRecordType *prt)
```

(continues on next page)

(continued from previous page)

```
{
    aoRecord *prec = 0;
    prt->papFldDes[aoRecordNAME]->size = sizeof(prec->name);
    ... code to compute size for remaining fields
    prt->papFldDes[aoRecordNAME]->offset = (char *)&prec->name - (char *)prec;
    ... code to compute offset for remaining fields
    prt->rec_size = sizeof(*prec);
    return 0;
}
epicsExportRegistrar(aoRecordSizeOffset);

#ifdef __cplusplus
}
#endif
#endif /* GEN_SIZE_OFFSET */

#endif /* INC_aoRecord_H */
```

The analog output record support module and all associated device support modules should include this file. No other code should use it.

Let's discuss the various parts of the file:

- The `enum` generated from the menu definition should be used to provide values for the field associated with that menu.
- The `typedef struct` defining the record are used by record support and device support to access the fields in an analog output record.
- The next `enum` defines an index number for each field within the record. This is useful for the record support routines that are passed a pointer to a `DBADDR` structure. They can have code like the following:

```
switch (dbGetFieldIndex(pdbAddr)) {
    case aoRecordVAL :
        ...
        break;
    case aoRecordXXX:
        ...
        break;
    default:
        ...
}
```

The generated routine `aoRecordSizeOffset` is executed when the record type gets registered with an IOC. The routine is compiled with the record type code, and is marked static so it will not be visible outside of that file. The associate record support source code MUST include the generated header file only after defining the `GEN_SIZE_OFFSET` macro like this:

```
#define GEN_SIZE_OFFSET
#include "aoRecord.h"
#undef GEN_SIZE_OFFSET
```

This convention ensures that the routine is defined exactly once. The `epicsExportRegistrar` statement ensures that the record registration code can find and call the routine.

dbdExpand.pl

```
dbdExpand.pl [-D] [-I dir] [-S mac=sub] [-o out.dbd] in.dbd ...
```

This program reads and combines the database definition from all the input files, then writes a single output file containing all information from the input files. The output content differs from the input in that comment lines are removed, and all defined macros and include files are expanded. Unlike the previous `dbExpand` program, this program does not understand database instances and cannot be used with `.db` or `.vdb` files.

Multiple `-I` options can be provided to specify directories that must be searched when looking for included files. Multiple `-S` options are allowed for macro substitution, or multiple macros can be specified within a single option. If no output filename is specified with the `-o out.dbd` option then the output will go to stdout.

The `-D` option causes the program to output Makefile dependency information for the output file to standard output, instead of actually performing the functions describe above.

dbLoadDatabase

```
dbLoadDatabase(char *dbdfile, char *path, char *substitutions)
```

This IOC command loads a database file which may contain any of the Database Definitions described in this chapter. The `dbdfile` string may contain environment variable macros of the form `${MOTOR}` which will be expanded before the file is opened. Both the `path` and `substitutions` parameters can be null or empty, and are usually ommitted. Note that `dbLoadDatabase` should only used to load Database Definition (`.dbd`) files, although it is currently possible to use it for loading Record Instance (`.db`) files as well.

As each line of the file is read, the substitutions specified in `substitutions` are performed. Substitutions are specified as follows:

```
"var1=sub1,var2=sub3,..."
```

Variables are used in the file with the syntax `$(var)` or `${var}`. If the substitution string

```
"a=1,b=2,c=\"this is a test\""
```

were used, any variables `$(a)`, `$(b)`, `$(c)` in the database file would have the appropriate values substituted during parsing.

dbLoadRecords

```
dbLoadRecords(char* dbfile, char* substitutions)
```

This IOC command loads a file containing record instances, record aliases and/or breakpoint tables. The `dbfile` string may contain environment variable macros of the form `${MOTOR}` which will be expanded before the file is opened. The `substitutions` parameter can be null or empty, and is often ommitted. Note that `dbLoadRecords` should only used to load Record Instance (`.db`) files, although it is currently possible to use it for loading Database Definition (`.dbd`) files as well.

Example

For example, let the file `test.db` contain:

```
record(ai, "$(pre)testrec1")
record(ai, "$(pre)testrec2")
record(stringout, "$(pre)testrec3") {
    field(VAL, "$(STR)")
    field(SCAN, "$(SCAN)")
}
```

Then issuing the command:

```
dbLoadRecords("test.db", "pre=TEST,STR=test,SCAN=Passive")
```

gives the same results as loading:

```
record(ai, "TESTtestrec1")
record(ai, "TESTtestrec2")
record(stringout, "TESTtestrec3") {
    field(VAL, "test")
    field(SCAN, "Passive")
}
```

dbLoadTemplate

```
dbLoadTemplate(char *subfile, char *substitutions)
```

This IOC command reads a template substitutions file which provides instructions for loading database instance files and gives values for the `$(xxx)` macros they may contain. This command performs those substitutions while loading the database instances requested.

The `subfile` parameter gives the name of the template substitution file to be used. The optional `substitutions` parameter may contain additional global macro values, which can be overridden by values given within the substitution file.

The MSI program can be used to expand templates at build-time instead of using this command at run-time; both understand the same substitution file syntax.

Template File Syntax

The template substitution file syntax is described in the following Extended Backus-Naur Form grammar:

```
substitution-file ::= ( global-defs | template-subs )+

global-defs ::= 'global' '{' variable-defs? '}'

template-subs ::= template-filename '{' subs? '}'
template-filename ::= 'file' file-name
subs ::= pattern-subs | variable-subs

pattern-subs ::= 'pattern' '{' pattern-names? '}' pattern-defs?
```

(continues on next page)

(continued from previous page)

```
pattern-names ::= ( variable-name ','? )+
pattern-defs ::= ( global-defs | ( '{' pattern-values? '}' ) )+
pattern-values ::= ( value ','? )+

variable-subs ::= ( global-defs | ( '{' variable-defs? '}' ) )+
variable-defs ::= ( variable-def ','? )+
variable-def ::= variable-name '=' value

variable-name ::= variable-name-start variable-name-char*
file-name ::= file-name-char+ | double-quoted-str | single-quoted-str
value ::= value-char+ | double-quoted-str | single-quoted-str

double-quoted-str ::= '"' (double-quoted-char | escaped-char)* '"'
single-quoted-str ::= "'" (single-quoted-char | escaped-char)* "'"
double-quoted-char ::= [^"\]
single-quoted-char ::= [^'\]
escaped-char ::= '\' .

value-char ::= [a-zA-Z0-9_+:;./\<>[] | '-' | ']'
variable-name-start ::= [a-zA-Z_]
variable-name-char ::= [a-zA-Z0-9_]
file-name-char ::= [a-zA-Z0-9_+:;./\] | '-'
```

Note that the current implementation may accept a wider range of characters for the last three definitions than those listed here, but future releases may restrict the characters to those given above.

Any record instance file names must appear inside quotation marks if the name contains any environment variable macros of the form `${ENV_VAR_NAME}`, which will be expanded before the named file is opened.

Template File Formats

Two different template formats are supported by the syntax rules given above. The format is either:

```
file name.template {
    { var1=sub1_for_set1, var2=sub2_for_set1, var3=sub3_for_set1, ... }
    { var1=sub1_for_set2, var2=sub2_for_set2, var3=sub3_for_set2, ... }
    { var1=sub1_for_set3, var2=sub2_for_set3, var3=sub3_for_set3, ... }
}
```

or:

```
file name.template {
pattern { var1, var2, var3, ... }
    { sub1_for_set1, sub2_for_set1, sub3_for_set1, ... }
    { sub1_for_set2, sub2_for_set2, sub3_for_set2, ... }
    { sub1_for_set3, sub2_for_set3, sub3_for_set3, ... }
}
```

The first line (`file name.template`) specifies the record instance input file. The file name may appear inside double quotation marks; these are required if the name contains any characters that are not in the following set, or if it contains environment variable macros of the form `${VAR_NAME}` which must be expanded to generate the file name:

```
a-z A-Z 0-9 _ + - . / \ : ; [ ] < >
```

Each set of definitions enclosed in {} is variable substitution for the input file. The input file has each set applied to it to produce one composite file with all the completed substitutions in it. Version 1 should be obvious. In version 2, the variables are listed in the `pattern{}` line, which must precede the braced substitution lines. The braced substitution lines contains sets which match up with the `pattern{}` line.

Example

Two simple template file examples are shown below. The examples specify the same substitutions to perform: `this=sub1` and `that=sub2` for a first set, and `this=sub3` and `that=sub4` for a second set.

```
file test.template {
    { this=sub1,that=sub2 }
    { this=sub3,that=sub4 }
}

file test.template {
    pattern{this,that}
    {sub1,sub2}
    {sub3,sub4 }
}
```

Assume that the file `test.template` contains:

```
record(ai,"$(this)record") {
    field(DESC,"this = $(this)")
}
record(ai,"$(that)record") {
    field(DESC,"this = $(that)")
}
```

Using `dbLoadTemplate` with either input is the same as defining the records:

```
record(ai,"sub1record") {
    field(DESC,"this = sub1")
}
record(ai,"sub2record") {
    field(DESC,"this = sub2")
}

record(ai,"sub3record") {
    field(DESC,"this = sub3")
}
record(ai,"sub4record") {
    field(DESC,"this = sub4")
}
```

1.13.5 IOC Initialization

Tags: developer

Table of Contents

Overview - Environments requiring a main program

If a main program is required (most likely on all environments except vxWorks and RTEMS), then initialization is performed by statements residing in startup scripts which are executed by iocsh. An example main program is:

```
int main(int argc,char  *argv[])
{
   if (argc >= 2) {
       iocsh(argv[1]);
       epicsThreadSleep(.2);
   }
   iocsh(NULL);
   epicsExit(0)
   return 0;
}
```

The first call to iocsh executes commands from the startup script filename which must be passed as an argument to the program. The second call to iocsh with a NULL argument puts iocsh into interactive mode. This allows the user to issue the commands described in the chapter on "IOC Test Facilities" as well as some additional commands like help.

The command file passed is usually called the startup script, and contains statements like these:

```
< envPaths
cd ${TOP}
dbLoadDatabase "dbd/appname.dbd"
appname_registerRecordDeviceDriver pdbbase
dbLoadRecords "db/file.db", "macro=value"
cd ${TOP}/iocBoot/${IOC}
iocInit
```

The envPaths file is automatically generated in the IOC's boot directory and defines several environment variables that are useful later in the startup script. The definitions shown below are always provided; additional entries will be created for each support module referenced in the application's configure/RELEASE file:

```
epicsEnvSet("ARCH","linux-x86")
epicsEnvSet("IOC","iocname")
epicsEnvSet("TOP","/path/to/application")
epicsEnvSet("EPICS_BASE","/path/to/base")
```

Overview - vxWorks

After vxWorks is loaded at IOC boot time, commands like the following, normally placed in the vxWorks startup script, are issued to load and initialize the application code:

```
# Many vxWorks board support packages need the following:
#cd <full path to IOC boot directory>
< cdCommands
cd topbin
ld 0,0, "appname.munch"

cd top
dbLoadDatabase "dbd/appname.dbd"
appname_registerRecordDeviceDriver pdbbase
```

(continues on next page)

(continued from previous page)

```
dbLoadRecords "db/file.db", "macro=value"

cd startup
iocInit
```

The cdCommands script is automatically generated in the IOC boot directory and defines several vxWorks global variables that allow cd commands to various locations, and also sets several environment variables. The definitions shown below are always provided; additional entries will be created for each support module referenced in the application's configure/RELEASE file:

```
startup = "/path/to/application/iocBoot/iocname"
putenv "ARCH=vxWorks-68040"
putenv "IOC=iocname"
top = "/path/to/application"
putenv "TOP=/path/to/application"
topbin = "/path/to/application/bin/vxWorks-68040"
epics_base = "/path/to/base"
putenv "EPICS_BASE=/path/to/base"
epics_basebin = "/path/to/base/bin/vxWorks-68040"
```

The **ld** command in the startup script loads EPICS core, the record, device and driver support the IOC needs, and any application specific modules that have been linked into it.

dbLoadDatabase loads database definition files describing the record/device/driver support used by the application..

dbLoadRecords loads record instance definitions.

iocInit initializes the various epics components and starts the IOC running.

Overview - RTEMS

RTEMS applications can start up in many different ways depending on the board-support package for a particular piece of hardware. Systems which use the Cexp package can be treated much like vxWorks. Other systems first read initialization parameters from non-volatile memory or from a BOOTP/DHCP server. The exact mechanism depends upon the BSP. TFTP or NFS filesystems are then mounted and the IOC shell is used to read commands from a startup script. The location of this startup script is specified by a initialization parameter. This script is often similar or identical to the one used with vxWorks. The RTEMS startup code calls

```
epicsRtemsInitPreSetBootConfigFromNVRAM(struct rtems_bsdnet_config  *);
```

just before setting the initialization parameters from non-volatile memory, and

```
epicsRtemsInitPostSetBootConfigFromNVRAM(struct rtems_bsdnet_config  *);
```

just after setting the initialization parameters. An application may provide either or both of these routines to perform any custom initialization required. These function prototypes and some useful external variable declarations can be found in the header file epicsRtemsInitHooks.h

IOC Initialization

An IOC is normally started with the **iocInit** command as shown in the startup scripts above, which is actually implemented in two distinct parts. The first part can be run separately as the iocBuild command, which puts the IOC into a quiescent state without allowing the various internal threads it starts to actually run. From this state the second command iocRun can be used to bring it online very quickly. A running IOC can be quiesced using the iocPause command, which freezes all internal operations; at this point the iocRun command can restart it from where it left off, or the IOC can be shut down (exit the program, or reboot on vxWorks/RTEMS). Most device support and drivers have not yet been written with the possibility of pausing an IOC in mind though, so this feature may not be safe to use on an IOC which talks to external devices or software.

IOC initialization using the iocBuild and iocRun commands then consists of the following steps:

Configure Main Thread

Provided the IOC has not already been initialized, initHookAtIocBuild is announced first.

The main thread's epicsThreadIsOkToBlock flag is set, the message "Starting iocInit" is logged and epicsSignalInstallSigHupIgnore called, which on Unix architectures prevents the process from shutting down if it later receives a HUP signal.

At this point, initHookAtBeginning is announced.

General Purpose Modules

Calls coreRelease which prints a message showing which version of iocCore is being run.

Calls taskwdInit to start the task watchdog. This accepts requests to watch other tasks. It runs periodically and checks to see if any of the tasks is suspended. If so it issues an error message, and can also invoke callback routines registered by the task itself or by other software that is interested in the state of the IOC. See "Task Watchdog" for details.

Starts the general purpose callback tasks by calling callbackInit. Three tasks are started at different scheduling priorities.

initHookAfterCallbackInit is announced.

Channel Access Links

Calls dbCaLinkInit. The initializes the module that handles database channel access links, but does not allow its task to run yet.

initHookAfterCaLinkInit is announced.

Driver Support

initDrvSup locates each device driver entry table and calls the init routine of each driver.

initHookAfterInitDrvSup is announced.

Record Support

initRecSup locates each record support entry table and calls the init routine for each record type.

initHookAfterInitRecSup is announced.

Device Support

initDevSup locates each device support entry table and calls its init routine specifying that this is the initial call.

initHookAfterInitDevSup is announced.

Database Records

initDatabase is called which makes three passes over the database performing the following functions:

1. Initializes the fields RSET, RDES, MLOK, MLIS, PACT and DSET for each record.

   Calls record support's init_record (first pass).
2. Convert each PV_LINK into a DB_LINK or CA_LINK

   Calls any extended device support's add_record routine.
3. Calls record support's init_record (second pass).

Finally it registers an epicsAtExit routine to shut down the database when the IOC application exits.

Next dbLockInitRecords is called to create the lock sets.

Then dbBkptInit is run to initialize the database debugging module.

initHookAfterInitDatabase is announced.

Device Support again

initDevSup locates each device support entry table and calls its init routine specifying that this is the final call.

initHookAfterFinishDevSup is announced.

Scanning and Access Security

The periodic, event, and I/O event scanners are initialized by calling scanInit, but the scan threads created are not allowed to process any records yet.

A call to asInit initailizes access security. If this reports failure, the IOC initialization is aborted.

dbProcessNotifyInit initializes support for process notification.

After a short delay to allow settling, initHookAfterScanInit is announced.

Initial Processing

initialProcess processes all records that have PINI set to YES.

initHookAfterInitialProcess is announced.

Channel Access Server

The Channel Access server is started by calling rsrv_init, but its tasks are not allowed to run so it does not announce its presence to the network yet.

initHookAfterCaServerInit is announced.

At this point, the IOC has been fully initialized but is still quiescent. initHookAfterIocBuilt is announced. If started using iocBuild this command completes here.

Enable Record Processing

If the iocRun command is used to bring the IOC out of its initial quiescent state, it starts here.

initHookAtIocRun is announced.

The routines scanRun and dbCaRun are called in turn to enable their associated tasks and set the global variable interruptAccept to TRUE (this now happens inside scanRun). Until this is set all I/O interrupts should have been ignored.

initHookAfterDatabaseRunning is announced. If the iocRun command (or iocInit) is being executed for the first time, initHookAfterInterruptAccept is announced.

Enable CA Server

The Channel Access server tasks are allowed to run by calling rsrv_run.

initHookAfterCaServerRunning is announced. If the IOC is starting for the first time, initHookAtEnd is announced.

A command completion message is logged, and initHookAfterIocRunning is announced.

Pausing an IOC

The command iocPause brings a running IOC to a quiescent state with all record processing frozen (other than possibly the completion of asynchronous I/O operations). A paused IOC may be able to be restarted using the iocRun command, but whether it will fully recover or not can depend on how long it has been quiescent and the status of any device drivers which have been running. The operations which make up the pause operation are as follows:

1. initHookAtIocPause is announced.
2. The Channel Access Server tasks are paused by calling rsrv_pause
3. initHookAfterCaServerPaused is announced.
4. The routines dbCaPause and scanPause are called to pause their associated tasks and set the global variable interruptAccept to FALSE.
5. initHookAfterDatabasePaused is announced.
6. After logging a pause message, initHookAfterIocPaused is announced.

Changing iocCore fixed limits

The following commands can be issued after iocCore is loaded to change iocCore fixed limits. The commands should be given before any dbLoadDatabase commands.

```
callbackSetQueueSize(size)
dbPvdTableSize(size)
scanOnceSetQueueSize(size)
errlogInit(buffersize)
errlogInit2(buffersize, maxMessageSize)
```

callbackSetQueueSize

Requests for the general purpose callback tasks are placed in a ring buffer. This command can be used to set the size for the ring buffers. The default is 2000. A message is issued when a ring buffer overflows. It should rarely be necessary to override this default. Normally the ring buffer overflow messages appear when a callback task fails.

dbPvdTableSize

Record instance names are stored in a process variable directory, which is a hash table. The default number of hash entries is 512. dbPvdTableSize can be called to change the size. It must be called before any dbLoad commands and must be a power of 2 between 256 and 65536. If an IOC contains very large databases (several thousand records) then a larger hash table size speeds up searches for records.

scanOnceSetQueueSize

scanOnce requests are placed in a ring buffer. This command can be used to set the size for the ring buffer. The default is 1000. It should rarely be necessary to override this default. Normally the ring buffer overflow messages appear when the scanOnce task fails.

errlogInit or errlogInit2

These commands can increase (but not decrease) the default buffer and maximum message sizes for the errlog message queue. The default buffer size is 1280 bytes, the maximum message size defaults to 256 bytes.

initHooks

The inithooks facility allows application functions to be called at various states during ioc initialization. The states are defined in initHooks.h, which contains the following definitions:

```
typedef enum {
    initHookAtIocBuild = 0,         / * Start of iocBuild/iocInit commands  */
    initHookAtBeginning,
    initHookAfterCallbackInit,
    initHookAfterCaLinkInit,
    initHookAfterInitDrvSup,
    initHookAfterInitRecSup,
    initHookAfterInitDevSup,
```

(continues on next page)

(continued from previous page)

```
    initHookAfterInitDatabase,
    initHookAfterFinishDevSup,
    initHookAfterScanInit,
    initHookAfterInitialProcess,
    initHookAfterCaServerInit,
    initHookAfterIocBuilt,          / * End of iocBuild command  */

    initHookAtIocRun,               / * Start of iocRun command  */
    initHookAfterDatabaseRunning,
    initHookAfterCaServerRunning,
    initHookAfterIocRunning,        / * End of iocRun/iocInit commands  */

    initHookAtIocPause,             / * Start of iocPause command  */
    initHookAfterCaServerPaused,
    initHookAfterDatabasePaused,
    initHookAfterIocPaused,         / * End of iocPause command  */

/ * Deprecated states, provided for backwards compatibility.
  * These states are announced at the same point they were before,
  * but will not be repeated if the IOC gets paused and restarted.
  */
    initHookAfterInterruptAccept,   / * After initHookAfterDatabaseRunning  */
    initHookAtEnd,                  / * Before initHookAfterIocRunning  */
}initHookState;

typedef void ( *initHookFunction)(initHookState state);
int initHookRegister(initHookFunction func);
const char  *initHookName(int state);
```

Any functions that are registered before iocInit reaches the desired state will be called when it reaches that state. The initHookName function returns a static string representation of the state passed into it which is intended for printing. The following skeleton code shows how to use this facility:

```
static initHookFunction myHookFunction;

int myHookInit(void)
{
  return(initHookRegister(myHookFunction));
}

static void myHookFunction(initHookState state)
{
  switch(state) {
    case initHookAfterInitRecSup:
      ...
      break;
    case initHookAfterInterruptAccept:
      ...
      break;
    default:
      break;
  }
```

(continues on next page)

(continued from previous page)

```
}
```

An arbitrary number of functions can be registered.

Environment Variables

Various environment variables are used by iocCore:

```
EPICS_CA_ADDR_LIST
EPICS_CA_AUTO_ADDR_LIST
EPICS_CA_CONN_TMO
EPICS_CAS_BEACON_PERIOD
EPICS_CA_REPEATER_PORT
EPICS_CA_SERVER_PORT
EPICS_CA_MAX_ARRAY_BYTES
EPICS_TS_NTP_INET
EPICS_IOC_LOG_PORT
EPICS_IOC_LOG_INET
```

For an explanation of the EPICS_CA_... and EPICS_CAS_... variables see the EPICS Channel Access Reference Manual. For an explanation of the EPICS_IOC_LOG_... variables see "iocLogClient" (To be added). EPICS_TS_NTP_INET is used only on vxWorks and RTEMS, where it sets the address of the Network Time Protocol server. If it is not defined the IOC uses the boot server as its NTP server.

These variables can be set through iocsh via the epicsEnvSet command, or on vxWorks using putenv. For example:

```
epicsEnvSet("EPICS_CA_CONN_TMO,"10")
```

All epicsEnvSet commands should be issued after iocCore is loaded and before any dbLoad commands.

The following commands can be issued to iocsh:

epicsPrtEnvParams - This shows just the environment variables used by iocCore.

epicsEnvShow - This shows all environment variables on your system.

Initialize Logging

Initialize the logging system. See the chapter on "IOC Error Logging" for details. The following can be used to direct the log client to use a specific host log server.

```
epicsEnvSet("EPICS_IOC_LOG_PORT", "<port>")
epicsEnvSet("EPICS_IOC_LOG_INET", "<inet addr>")
```

These command must be given immediately after iocCore is loaded.

To start logging you must issue the command:

```
iocLogInit
```

1.13.6 IOC Access Security

Tags: developer advanced

Table of Contents

Features

Access security protects IOC databases from unauthorized Channel Access or pvAccess Clients. Access security is based on the following:

Who
: User id of the client(Channel Access/pvAccess).

Where
: Host id where the user is logged on. This is the host on which the client exists. Thus no attempt is made to see if a user is local or is remotely logged on to the host.

What
: Individual fields of records are protected. Each record has a field containing the Access Security Group (ASG) to which the record belongs. Each field has an access security level, either ASL0 or ASL1. The security level is defined in the record definition file (.dbd). Thus the access security level for a field is the same for all record instances of a record type.

When
: Access rules can contain input links and calculations similar to the calculation record.

Limitations

An IOC database can be accessed only via pvAccess, Channel Access or the ioc (or vxWorks) shell. It is assumed that access to the local IOC console is protected via physical security, and that network access is protected via normal networking and physical security methods.

No attempt has been made to protect against the sophisticated saboteur. Network and physical security methods must be used to limit access to the subnet on which the IOCs reside.

Definitions

This document uses the following terms:

ASL
Access Security Level.

ASG
Access Security Group

UAG
User Access Group

HAG
Host Access Group

Quick Start

In order to "turn on" access security for a particular IOC the following must be done:

- Create the access security file.
- IOC databases may have to be modified
  - Record instances may have to have values assigned to field ASG. If ASG is null the record is in group DEFAULT.
  - Access security files can be reloaded after iocInit via a subroutine record with asSubInit and asSubProcess as the associated subroutines. Writing the value 1 to this record will cause a reload.
  - The startup script must contain the following command before iocInit.

```
asSetFilename("/full/path/to/accessSecurityFile")
```

- The following is an optional command.

```
asSetSubstitutions("var1=sub1,var2=sub2,...")
```

The following rules decide if access security is turned on for an IOC:

- If asSetFilename is not executed before iocInit, access security will never be started.
- If asSetFile is given and any error occurs while first initializing access security, then all access to that ioc is denied.
- If after successfully starting access security, an attempt is made to restart and an error occurs then the previous access security configuration is maintained.

After an IOC has been booted with access security enabled, the access security rules can be changed by issuing the asSetFilename, asSetSubstitutions, and asInit. The functions asInitialize, asInitFile, and asInitFP, which are described below, can also be used.

Access Security Configuration File

This section describes the format of a file containing definitions of the user access groups, host access groups, and access security groups. An IOC creates an access configuration database by reading an access configuration file (the extension .acf is recommended). Lets first give a simple example and then a complete description of the syntax.

Simple Example

```
UAG(uag) {user1,user2}
HAG(hag) {host1,host2}
ASG(DEFAULT) {
        RULE(1,READ)
        RULE(1,WRITE) {
                UAG(uag)
                HAG(hag)
       }
}
```

These rules provide read access to anyone located anywhere and write access to user1 and user2 if they are located at host1 or host2.

Syntax Definition

In the following description:

[ ] surrounds optional elements

| separates alternatives

... means that an arbitrary number of definitions may be given.

introduces a comment line

The elements <name>, <user>, <host>, <pvname> and <calculation> can be given as quoted or unquoted strings. The rules for unquoted strings are the same as for database definitions.

```
UAG(<name>) [{ <user> [, <user> ...] }]
...
HAG(<name>) [{ <host> [, <host> ...] }]
...
ASG(<name>) [{
    [INP<index>(<pvname>)
    ...]
    RULE(<level>,NONE | READ | WRITE [, NOTRAPWRITE | TRAPWRITE]) {
        [UAG(<name> [,<name> ...])]
        [HAG(<name> [,<name> ...])]
        CALC(<calculation>)
    }
    ...
}]
...
```

Discussion

- UAG: User Access Group. This is a list of user names. The list may be empty. A user name may appear in more than one UAG. To match, a user name must be identical to the user name read by the CA client library running on the client machine. For vxWorks clients, the user name is usually taken from the user field of the boot parameters.
- HAG: Host Access Group. This is a list of host names. It may be empty. The same host name can appear in multiple HAGs. To match, a host name must match the host name read by the CA client library running on the client machine; both names are converted to lower case before comparison however. For vxWorks clients, the host name is usually taken from the target name of the boot parameters.
- ASG: An access security group. The group DEFAULT is a special case. If a member specifies a null group or a group which has no ASG definition then the member is assigned to the group DEFAULT.
- INP<index>Index must have one of the values A to L. These are just like the INP fields of a calculation record. It is necessary to define INP fields if a CALC field is defined in any RULE for the ASG.
- RULE This defines access permissions. <level> must be 0 or 1. Permission for a level 1 field implies permission for level 0 fields. The permissions are NONE, READ, and WRITE. WRITE permission implies READ permission. The standard EPICS record types have all fields set to level 1 except for VAL, CMD (command), and RES (reset). An optional argument specifies if writes should be trapped. See the section below on trapping Channel Access writes for how this is used. If not given the default is NOTRAPWRITE.
  - UAG specifies a list of user access groups that can have the access privilege. If UAG is not defined then all users are allowed.
  - HAG specifies a list of host access groups that have the access privilege. If HAG is not defined then all hosts are allowed.
  - CALC is just like the CALC field of a calculation record except that the result must evaluate to TRUE or FALSE. The rule only applies if the calculation result is TRUE, where the actual test for TRUE is (0.99 < result < 1.01). Anything else is regarded as FALSE and will cause the rule to be ignored. Assignment statements are not permitted in CALC expressions here.

Each IOC record contains a field ASG, which specifies the name of the ASG to which the record belongs. If this field is null or specifies a group which is not defined in the access security file then the record is placed in group DEFAULT.

The access privilege for a channel access client is determined as follows:

1. The ASG associated with the record is searched.
2. Each RULE is checked for the following:
   1. The field's level must be less than or equal to the level for this RULE.
   2. If UAG is defined, the user must belong to one of the specified UAGs. If UAG is not defined all users are accepted.
   3. If HAG is defined, the user's host must belong to one one of the HAGs. If HAG is not defined all hosts are accepted.
   4. If CALC is specified, the calculation must yield the value 1, i.e. TRUE. If any of the INP fields associated with this calculation are in INVALID alarm severity the calculation is considered false. The actual test for TRUE is .99 <result <1.01.
3. The maximum access allowed by step 2 is the access chosen.

Multiple RULEs can be defined for a given ASG, even RULEs with identical levels and access permissions. The TRAPWRITE setting used for a client is determined by the first WRITE rule that passes the rule checks.

ascheck - Check Syntax of Access Configuration File

After creating or modifying an access configuration file it can be checked for syntax errors by issuing the command:

```
ascheck -S "xxx=yyy,..." < "filename"
```

This is a Unix command. It displays errors on stdout. If no errors are detected it prints nothing. Only syntax errors not logic errors are detected. Thus it is still possible to get your self in trouble. The flag -S means a set of macro substitutions may appear. This is just like the macro substitutions for dbLoadDatabase.

IOC Access Security Initialization

In order to have access security turned on during IOC initialization the following command must appear in the startup file before iocInit is called:

```
asSetFilename("/full/path/to/access/security/file.acf")
```

If this command is not used then access security will not be started by iocInit. If an error occurs when iocInit calls asInit than all access to the ioc is disabled, i.e. no channel access client will be able to access the ioc. Note that this command does not read the file itself, it just saves the argument string for use later on, nor does it save the current working directory, which is why the use of an absolute path-name for the file is recommended (a path name could be specified relative to the current directory at the time when iocInit is run, but this is not recommended if the IOC also loads the subroutine record support as a later reload of the file might happen after the current directory had been changed).

Access security also supports macro substitution just like dbLoadDatabase. The following command specifies the desired substitutions:

```
asSetSubstitutions("var1=sub1,var2=sub2,...")
```

This command must be issued before iocInit.

After an IOC is initialized the access security database can be changed. The preferred way is via the subroutine record described in the next section. It can also be changed by issuing the following command to the vxWorks shell:

```
asInit
```

It is also possible to reissue asSetFilename and/or asSetSubstitutions before asInit. If any error occurs during asInit the old access security configuration is maintained. It is NOT permissible to call asInit before iocInit is called.

Restarting access security after ioc initialization is an expensive operation and should not be used as a regular procedure.

Database Configuration

Access Security Group

Each database record has a field ASG which holds a character string. Any database configuration tool can be used to give a value to this field. If the ASG of a record is not defined or is not equal to a ASG in the configuration file then the record is placed in DEFAULT.

Subroutine Record Support

Two subroutines, which can be attached to a subroutine record, are available (provided with iocCore):

```
asSubInit
asSubProcess
```

NOTE: These subroutines are automatically registered thus do NOT put a registrar definition in your database definition file.

If a record is created that attaches to these routines, it can be used to force the IOC to load a new access configuration database. To change the access configuration:

1. Modify the file specified by the last call to asSetFilename so that it contains the new configuration desired.
2. Write a 1 to the subroutine record VAL field. Note that this can be done via channel access.

The following action is taken:

1. When the value is found to be 1, asInit is called and the value set back to 0.
2. The record is treated as an asynchronous record. Completion occurs when the new access configuration has been initialized or a time-out occurs. If initialization fails the record is placed into alarm with a severity determined by BRSV.

Record Type Description

Each field of each record type has an associated access security level of ASL0 or ASL1 (default value). Fields which operators normally change are assigned ASL0, other fields are assigned ASL1. For example, the VAL field of an analog output record is assigned ASL0 and all other fields ASL1. This is because only the VAL field should be modified during normal operations.

Example:

Lets design a set of rules for a Linac. Assume the following:

1. Anyone can have read access to all fields at anytime.
2. Linac engineers, located in the injection control or control room, can have write access to most level 0 fields only if the Linac is not in operational mode.
3. Operators, located in the injection control or control room, can have write access to most level 0 fields anytime.
4. The operations supervisor, linac supervisor, and the application developers can have write access to all fields but must have some way of not changing something inadvertently.
5. Most records use the above rules but a few (high voltage power supplies, etc.) are placed under tighter control. These will follow rules 1 and 4 but not 2 or 3.
6. IOC channel access clients always have level 1 write privilege.

Most Linac IOC records will not have the ASG field defined and will thus be placed in ASG DEFAULT. The following records will have an ASG defined:

- LI:OPSTATE and any other records that need tighter control have ASG=”critical”. One such record could be a subroutine record used to cause a new access configuration file to be loaded. LI:OPSTATE has the value (0,1) if the Linac is (not operational, operational).

- LI:lev1permit has ASG="permit". In order for the opSup, linacSup, or an appDev to have write privilege to everything this record must be set to the value 1.

The following access configuration satisfies the above rules.

```
UAG(op) {op1,op2,superguy}
UAG(opSup) {superguy}
UAG(linac) {waw,nassiri,grelick,berg,fuja,gsm}
UAG(linacSup) {gsm}
UAG(appDev) {nda,kko}
HAG(icr) {silver,phebos,gaea}
HAG(cr) {mars,hera,gold}
HAG(ioc) {ioclic1,ioclic2,ioclid1,ioclid2,ioclid3,ioclid4,ioclid5}
ASG(DEFAULT) {
    INPA(LI:OPSTATE)
    INPB(LI:lev1permit)
    RULE(0,WRITE) {
        UAG(op)
        HAG(icr,cr)
        CALC("A=1")
    }
    RULE(0,WRITE) {
        UAG(op,linac,appdev)
        HAG(icr,cr)
        CALC("A=0")
    }
    RULE(1,WRITE) {
        UAG(opSup,linacSup,appdev)
        CALC("B=1")
    }
    RULE(1,READ)
    RULE(1,WRITE) {
        HAG(ioc)
    }
}
ASG(permit) {
    RULE(0,WRITE) {
        UAG(opSup,linacSup,appDev)
    }
    RULE(1,READ)
    RULE(1,WRITE) {
        HAG(ioc)
    }
}
ASG(critical) {
    INPB(LI:lev1permit)
    RULE(1,WRITE) {
        UAG(opSup,linacSup,appdev)
        CALC("B=1")
    }
    RULE(1,READ)
    RULE(1,WRITE) {
        HAG(ioc)
    }
```

(continues on next page)

(continued from previous page)

```
}
```

Summary of Functional Requirements

A brief summary of the Functional Requirements is:

1. Each field of each record type is assigned an access security level.
2. Each record instance is assigned to a unique access security group.
3. Each user is assigned to one or more user access groups.
4. Each node is assigned to a host access group.
5. For each access security group a set of access rules can be defined. Each rule specifies:
    1. Access security level
    2. READ or READ/WRITE access.
    3. An optional list of User Access Groups or * meaning anyone.
    4. An optional list of Host Access Groups or * meaning anywhere.
    5. Conditions based on values of process variables

Additional Requirements

Performance

Although the functional requirements do not mention it, a fundamental goal is performance. The design provides almost no overhead during normal database access and moderate overhead for the following: channel access client/server connection, ioc initialization, a change in value of a process variable referenced by an access calculation, and dynamically changing a records access control group. Dynamically changing the user access groups, host access groups, or the rules, however, can be a time consuming operation. This is done, however, by a low priority IOC task and thus does not impact normal ioc operation.

Generic Implementation

Access security should be implemented as a stand alone system, i.e. it should not be embedded tightly in database or channel access.

No Access Security within an IOC

No access security is invoked within an IOC . This means that database links and local channel access clients calls are not subject to access control. Also test routines such as dbgf should not be subject to access control.

Defaults

It must be possible to easily define default access rules.

Access Security is Optional

When an IOC is initialized, access security is optional.

pvAccess (QSRV) Specific Features

QSRV will enforce the access control policy loaded by the usual means (cf. asSetFilename()). This policy is applied to both Single and Group PVs. With Group PVs, restrictions are not defined for the group, but rather for the individual member records. The same policy will be applied regardless of how a record is accessed (individually, or through a group).

Policy application differs from CA (RSRV) in several ways:

Client hostname is always the numeric IP address. HAG() entries must either contain numeric IP addresses, or **asCheckClientIP=1** flag must be set to translate hostnames into IPs on ACF file load (effects CA server as well). This prevents clients from trivially forging "hostname". In additional to client usernames, UAG definitions may contained items beginning with "role/" which are matched against the list of groups of which the client username is a member. Username to group lookup is done internally to QSRV, and depends on IOC host authentication configuration. Note that this is still based on the client provided username string.

```
UAG(special) {
   someone, "role/op"
}
```

The "special" UAG will match CA or PVA clients with the username "someone". It will also match a PVA client if the client provided username is a member of the "op" group (supported on POSIX targets and Windows).

1.13.7 IOC Test Facilities

Tags: user developer

Table of Contents

- *dbDumpBreaktable*
- *dbPvdDump*

Overview

This chapter describes a number of IOC test routines that are of interest to both application developers and system developers. The routines are available from either iocsh or the vxWorks shell. In both shells the parentheses around arguments are optional. On vxWorks all character string arguments must be enclosed in double quote characters "" and all arguments must be separated by commas. For iocsh single or double quotes must be used around string arguments that contain spaces or commas but are otherwise optional, and arguments may be separated by either commas or spaces. For example:

```
dbpf("aiTest","2")
dbpf "aiTest","2"
```

are both valid with both iocsh and with the vxWorks shell.

```
dbpf aiTest 2
```

Is valid for iocsh but not for the vxWorks shell.

Both iosch and vxWorks shells allow output redirection, i.e. the standard output of any command can be redirected to a file. For example

```
dbl > dbl.lst
```

will send the output of the `dbl` command to the file `dbl.lst`

If iocsh is being used it provides help for all commands that have been registered. Just type

```
help
```

or

```
help pattern*
```

Database List, Get, Put

dbl

Database List:

```
dbl("<record type>","<field list>")
```

Examples

```
dbl
dbl("ai")
dbl("*")
dbl("")
```

This command prints the names of records in the run time database. If <record type> is empty (""), "*", or not specified, all records are listed. If <record type> is specified, then only the names of the records of that type are listed.

If <field list> is given and not empty then the values of the fields specified are also printed.

dbgrep

List Record Names That Match a Pattern:

```
dbgrep("<pattern>")
```

Examples

```
dbgrep("S0*")
dbgrep("*gpibAi*")
```

Lists all record names that match a pattern. The pattern can contain any characters that are legal in record names as well as "*", which matches 0 or more characters.

dbla

List Record Alias Names with optional pattern:

```
dbla
dbla("<pattern>")
```

Lists the names of all aliases (which match the pattern if given) and the records they refer to. Examples:

```
dbla
dbla "alia*"
```

dba

Database Address:

```
dba("<record_name.field_name>")
```

Example

```
dba("aitest")
dba("aitest.VAL")
```

This command calls dbNameToAddr and then prints the value of each field in the dbAddr structure describing the field. If the field name is not specified then VAL is assumed (the two examples above are equivalent).

dbgf

Get Field:

```
dbgf("<record_name.field_name>")
```

Example:

```
dbgf("aitest")
dbgf("aitest.VAL")
```

This performs a `dbNameToAddr` and then a `dbGetField`. It prints the field type and value. If the field name is not specified then `VAL` is assumed (the two examples above are equivalent). Note that `dbGetField` locks the record lockset, so `dbgf` will not work on a record with a stuck lockset; use `dbpr` instead in this case.

dbpf

Put Field:

```
dbpf("<record_name.field_name>","<value>")
```

Example:

```
dbpf("aitest","5.0")
```

This command performs a `dbNameToAddr` followed by a `dbPutField` and `dbgf`. If `<field_name>` is not specified `VAL` is assumed.

dbpr

Print Record:

```
dbpr("<record_name>",<interest level>)
```

Example

```
dbpr("aitest",2)
```

This command prints all fields of the specified record up to and including those with the indicated interest level. Interest level has one of the following values:

- 0: Fields of interest to an Application developer and that can be changed as a result of record processing.
- 1: Fields of interest to an Application developer and that do not change during record processing.
- 2: Fields of major interest to a System developer.
- 3: Fields of minor interest to a System developer.
- 4: Fields of no interest.

dbtr

Test Record:

```
dbtr("<record_name>")
```

This calls `dbNameToAddr`, then `dbProcess` and finally `dbpr` (interest level 3). Its purpose is to test record processing.

dbnr

Print number of records:

```
dbnr(<all_recordtypes>)
```

This command displays the number of records of each type and the total number of records. If `all_record_types` is 0 then only record types with record instances are displayed otherwise all record types are displayed.

Breakpoints

A breakpoint facility that allows the user to step through database processing on a per lockset basis. This facility has been constructed in such a way that the execution of all locksets other than ones with breakpoints will not be interrupted. This was done by executing the records in the context of a separate task.

The breakpoint facility records all attempts to process records in a lockset containing breakpoints. A record that is processed through external means, e.g.: a scan task, is called an entrypoint into that lockset. The `dbstat` command described below will list all detected entrypoints to a lockset, and at what rate they have been detected.

dbb

Set Breakpoint:

```
dbb("<record_name>")
```

Sets a breakpoint in a record. Automatically spawns the `bkptCont`, or breakpoint continuation task (one per lockset). Further record execution in this lockset is run within this task's context. This task will automatically quit if two conditions are met, all breakpoints have been removed from records within the lockset, and all breakpoints within the lockset have been continued.

dbd

Remove Breakpoint:

```
dbd("<record_name>")
```

Removes a breakpoint from a record.

dbs

Single Step:

```
dbs("<record_name>")
```

Steps through execution of records within a lockset. If this command is called without an argument, it will automatically step starting with the last detected breakpoint.

dbc

Continue:

```
dbc("<record_name>")
```

Continues execution until another breakpoint is found. This command may also be called without an argument.

dbp

Print Fields Of Suspended Record:

```
dbp("<record_name>,<interest_level>)
```

Prints out the fields of the last record whose execution was suspended.

dbap

Auto Print:

```
dbap("<record_name>")
```

Toggles the automatic record printing feature. If this feature is enabled for a given record, it will automatically be printed after the record is processed.

dbstat

Status:

```
dbstat
```

Prints out the status of all locksets that are suspended or contain breakpoints. This lists all the records with breakpoints set, what records have the autoprint feature set (by dbap), and what entrypoints have been detected. It also displays the vxWorks task ID of the breakpoint continuation task for the lockset. Here is an example output from this call:

```
LSet: 00009  Stopped at: so#B: 00001   T: 0x23cafac
             Entrypoint: so#C: 00001   C/S:     0.1
             Breakpoint: so(ap)
LSet: 00008#B: 00001   T: 0x22fee4c
             Breakpoint: output
```

The above indicates that two locksets contain breakpoints. One lockset is stopped at record "so." The other is not currently stopped, but contains a breakpoint at record "output." "LSet:" is the lockset number that is being considered. "#B:" is the number of breakpoints set in records within that lockset. "T:" is the vxWorks task ID of the continuation task. "C:" is the total number of calls to the entrypoint that have been detected. "C/S:" is the number of those calls that have been detected per second. (ap) indicates that the autoprint feature has been turned on for record "so."

Trace Processing

The user should also be aware of the field TPRO, which is present in every database record. If it is set TRUE then a message is printed each time its record is processed and a message is printed for each record processed as a result of it being processed.

Error Logging

eltc

Display error log messages on console:

```
eltc(int noYes)
```

This determines if error messages are displayed on the IOC console. 0 means no and any other value means yes.

errlogInit, errlogInit2

Initialize error log client buffering

```
errlogInit(int bufSize)
errlogInit2(int bufSize, int maxMsgSize)
```

The error log client maintains a circular buffer of messages that are waiting to be sent to the log server. If not set using one or other of these routines the default value for bufSize is 1280 bytes and for maxMsgSize is 256 bytes.

errlog

Send a message to the log server

```
errlog("<message>")
```

This command is provided for use from the ioc shell only. It sends its string argument and a new-line to the log server, without displaying it on the IOC console. Note that the iocsh will have expanded any environment variable macros in the string (if it was double-quoted) before passing it to errlog.

Hardware Reports

dbior

I/O Report:

```
dbior ("<driver_name>",<interest level>)
```

This command calls the report entry of the indicated driver. If <driver_name> is "" or *, then a report for all drivers is generated. The command also calls the report entry of all device support modules. Interest level is one of the following:

- 0: Print a short report for each module.
- 1: Print additional information.
- 2: Print even more info. The user may be prompted for options.

dbhcr

Hardware Configuration Report:

```
dbhcr()
```

This command produces a report of all hardware links. To use it on the IOC, issue the command:

```
dbhcr > report
```

The report will probably not be in the sort order desired. The Unix command:

```
sort report > report.sort
```

should produce the sort order you desire.

Scan Reports

scanppl

Print Periodic Lists:

```
scanppl(double rate)
```

This routine prints a list of all records in the periodic scan list of the specified rate. If rate is 0.0 all period lists are shown.

scanpel

Print Event Lists:

```
scanpel(int event_number)
```

This routine prints a list of all records in the event scan list for the specified event nunber. If event_number is 0 all event scan lists are shown.

scanpiol

Print I/O Event Lists:

```
scanpiol
```

This routine prints a list of all records in the I/O event scan lists.

General Time

The built-in time providers depend on the IOC's target architecture, so some of the specific subsystem report commands listed below are only available on the architectures that use that particular provider.

generalTimeReport

Format:

```
generalTimeReport(int level)
```

This routine displays the time providers and their priority levels that have registered with the General Time subsystem for both current and event times. At level 1 it also shows the current time as obtained from each provider.

installLastResortEventProvider

Format:

```
installLastResortEventProvider
```

Installs the optional Last Resort event provider at priority 999, which returns the current time for every event number.

NTPTime_Report

Format:

```
NTPTime_Report(int level)
```

Only vxWorks and RTEMS targets use this time provider. The report displays the provider's synchronization state, and at interest level 1 it also gives the synchronization interval, when it last synchronized, the nominal and measured system tick rates, and on vxWorks the NTP server address.

NTPTime_Shutdown

Format:

```
NTPTime_Shutdown
```

On vxWorks and RTEMS this command shuts down the NTP time synchronization thread. With the thread shut down, the driver will no longer act as a current time provider.

ClockTime_Report

Format:

```
ClockTime_Report(int level)
```

This time provider is used on several target architectures, registered as the time provider of last resort. On vxWorks and RTEMS the report displays the synchronization state, when it last synchronized the system time with a higher priority provider, and the synchronization interval. On workstation operating systems the synchronization task is not started on the assumption that some other process is taking care of synchronzing the OS clock as appropriate, so the report is minimal.

ClockTime_Shutdown

Format:

```
ClockTime_Shutdown
```

Some sites may prefer to provide their own implementation of a system clock time provider to replace the built-in one. On vxWorks and RTEMS this command stops the OS Clock synchronization thread, allowing the OS clock to free-run. The time provider *will* continue to return the current system time after this command is used however.

Access Security Commands

asSetSubstitutions

Format:

```
asSetSubstitutions("substitutions")
```

Specifies macro substitutions used when access security is initialized.

asSetFilename

Format:

```
asSetFilename("<filename>")
```

This command defines a new access security file.

asInit

Format:

```
asInit
```

This command reinitializes the access security system. It rereads the access security file in order to create the new access security database. This command is useful either because the `asSetFilename` command was used to change the file or because the file itself was modified. Note that it is also possible to reinitialize the access security via a subroutine record. See the access security document for details.

asdbdump

Format:

```
asdbdump
```

This provides a complete dump of the access security database.

aspuag

Format:

```
aspuag("<user access group>")
```

Print the members of the user access group. If no user access group is specified then the members of all user access groups are displayed.

asphag

Format:

```
asphag("<host access group>")
```

Print the members of the host access group. If no host access group is specified then the members of all host access groups are displayed.

asprules

Format:

```
asprules("<access security group>")
```

Print the rules for the specified access security group or if no group is specified for all groups.

aspmem

Format:

```
aspmem("<access security group>", <print clients>)
```

Print the members (records) that belong to the specified access security group, for all groups if no group is specified. If <print clients> is (0, 1) then Channel Access clients attached to each member (are not, are) shown.

Channel Access Reports

casr

Channel Access Server Report

```
casr(<level>)
```

Level can have one of the following values:

0

Prints server's protocol version level and a one line summary for each client attached. The summary lines contain the client's login name, client's host name, client's protocol version number, and the number of channel created within the server by the client.

1

Level one provides all information in level 0 and adds the task id used by the server for each client, the client's IP protocol type, the file number used by the server for the client, the number of seconds elapsed since the last request was received from the client, the number of seconds elapsed since the last response was sent to the client, the number of unprocessed request bytes from the client, the number of response bytes which have not been flushed to the client, the client's IP address, the client's port number, and the client's state.

2

Level two provides all information in levels 0 and 1 and adds the number of bytes allocated by each client and a list of channel names used by each client. Level 2 also provides information about the number of bytes in the server's free memory pool, the distribution of entries in the server's resource hash table, and the list of IP addresses to which the server is sending beacons. The channel names are shown in the form:

<name>(nrw)

where

n is number of ca_add_events the client has on this channel

r is (-,R) if client (does not, does) have read access to the channel.

w is(-, W) if client (does not, does) have write access to the channel.

dbel

Format:

```
dbel("<record_name>")
```

This routine prints the Channel Access event list for the specified record.

dbcar

Database to Channel Access Report - See "Record Link Reports"

ascar

Format:

```
ascar(level)
```

Prints a report of the channel access links for the INP fields of the access security rules. Level 0 produces a summary report. Level 1 produces a summary report plus details on any unconnect channels. Level 2 produces the summary nreport plus a detail report on each channel.

Interrupt Vectors

veclist

Format:

```
veclist
```

NOTE: This routine is only available on vxWorks. On PowerPC CPUs it requires BSP support to work, and even then it cannot display chained interrupts using the same vector.

Print Interrupt Vector List

Miscellaneous

epicsParamShow

Format:

```
epicsParamShow
```

or

```
epicsPrtEnvParams
```

Print the environment variables that are created with epicsEnvSet. These are defined in <base>/config/CONFIG_ENV and <base>/config/CONFIG_SITE_ENV or else by user applications calling `epicsEnvSet`.

epicsEnvShow

Format:

```
epicsEnvShow("<name>")
```

Show Environment variables. On vxWorks it shows the variables created via calls to `putenv`.

coreRelease

Format:

```
coreRelease
```

Print release information for iocCore.

Database System Test Routines

These routines are normally only of interest to EPICS system developers NOT to Application Developers.

dbtgf

Test Get Field:

```
dbtgf("<record_name.field_name>")
```

Example:

```
dbtgf("aitest")
dbtgf("aitest.VAL")
```

This performs a `dbNameToAddr` and then calls `dbGetField` with all possible request types and options. It prints the results of each call. This routine is of most interest to system developers for testing database access.

dbtpf

Test Put Field:

```
dbtpf("<record_name.field_name>","<value>")
```

Example:

```
dbtpf("aitest","5.0")
```

This command performs a `dbNameToAddr`, then calls `dbPutField`, followed by `dbgf` for each possible request type. This routine is of interest to system developers for testing database access.

dbtpn

Test Process Notify:

```
dbtpn("<record_name.field_name>")
dbtpn("<record_name.field_name>","<value>")
```

Example:

```
dbtpn("aitest")
dbtpn("aitest","5.0")
```

This command performs a `dbProcessNotify` request. If a non-null value argument string is provided it issues a `putProcessRequest` to the named record; if no value is provided it issues a `processGetRequest`. This routine is mainly of interest to system developers for testing database access.

Record Link Reports

dblsr

Lock Set Report:

```
dblsr(<recordname>,<level>)
```

This command generates a report showing the lock set to which each record belongs. If `recordname` is 0, "", or "*" all records are shown, otherwise only records in the same lock set as `recordname` are shown.

`level` can have the following values:

0 - Show lock set information only.

1 - Show each record in the lock set.

2 - Show each record and all database links in the lock set.

dbLockShowLocked

Show locked locksets:

```
dbLockShowLocked(<level>)
```

This command generates a report showing all locked locksets, the records they contain, the lockset state and the thread that currently owns the lockset. The `level` argument is passed to `epicsMutexShow` to adjust the information reported about each locked epicsMutex.

dbcar

Database to channel access report

```
dbcar(<recordname>,<level>)
```

This command generates a report showing database channel access links. If `recordname` is "*" then information about all records is shown otherwise only information about the specified record.

`level` can have the following values:

0 - Show summary information only.

1 - Show summary and each CA link that is not connected.

2 - Show summary and status of each CA link.

dbhcr

Report hardware links. See "Hardware Reports".

Old Database Access Testing

These routines are of interest to EPICS system developers. They are used to test the old database access interface, which is still used by Channel Access.

gft

Get Field Test:

```
gft("<record_name.field_name>")
```

Example:

```
gft("aitest")
gft("aitest.VAL")
```

This performs a `db_name_to_addr` and then calls `db_get_field` with all possible request types. It prints the results of each call. This routine is of interest to system developers for testing database access.

pft

Put Field Test:

```
pft("<record_name.field_name>","<value>")
```

Example:

```
pft("aitest","5.0")
```

This command performs a `db_name_to_addr`, `db_put_field`, `db_get_field` and prints the result for each possible request type. This routine is of interest to system developers for testing database access.

tpn

Test Process Notify:

```
tpn("<record_name.field_name>","<value>")
```

Example:

```
tpn("aitest","5.0")
```

This routine tests the `dbProcessNotify` API when used via the old database access interface. It only supports issuing a `putProcessRequest` to the named record.

Routines to dump database information

dbDumpPath

Dump Path:

```
dbDumpPath(pdbbase)
```

Example:

```
dbDumpPath(pdbbase)
```

The current path for database includes is displayed.

dbDumpMenu

Dump Menu:

```
dbDumpMenu(pdbbase,"<menu>")
```

Example:

```
dbDumpMenu(pdbbase,"menuScan")
```

If the second argument is 0 then all menus are displayed.

dbDumpRecordType

Dump Record Description:

```
dbDumpRecordType(pdbbase,"<record type>")
```

Example:

```
dbDumpRecordType(pdbbase,"ai")
```

If the second argument is 0 then all descriptions of all records are displayed.

dbDumpField

Dump Field Description:

```
dbDumpField(pdbbase,"<record type>","<field name>")
```

Example:

```
dbDumpField(pdbbase,"ai","VAL")
```

If the second argument is 0 then the field descriptions of all records are displayed. If the third argument is 0 then the description of all fields are displayed.

dbDumpDevice

Dump Device Support:

```
dbDumpDevice(pdbbase,"<record type>")
```

Example:

```
dbDumpDevice(pdbbase,"ai")
```

If the second argument is 0 then the device support for all record types is displayed.

dbDumpDriver

Dump Driver Support:

```
dbDumpDriver(pdbbase)
```

Example:

```
dbDumpDriver(pdbbase)
```

dbDumpRecord

Dump Record Instances:

```
dbDumpRecord(pdbbase,"<record type>",level)
```

Example:

```
dbDumpRecords(pdbbase,"ai")
```

If the second argument is 0 then the record instances for all record types are displayed. The third argument determines which fields are displayed just like for the command `dbpr`.

dbDumpBreaktable

Dump breakpoint table

```
dbDumpBreaktable(pdbbase,name)
```

Example:

```
dbDumpBreaktable(pdbbase,"typeKdegF")
```

This command dumps a breakpoint table. If the second argument is 0 all breakpoint tables are dumped.

dbPvdDump

Dump the Process variable Directory:

```
dbPvdDump(pdbbase,verbose)
```

Example:

```
dbPvdDump(pdbbase,0)
```

This command shows how many records are mapped to each hash table entry of the process variable directory. If verbose is not 0 then the command also displays the names which hash to each hash table entry.

1.14 How to Add a New Breakpoint Table

Tags: developer advanced

1. Copy `menuConvert.dbd` from `base/dbd` to the app's src directory.
2. In the src directory, create a breakpoint table file `<bpname>.dbd`. Look at `base/dbd/bpt*.dbd` for the proper format.
3. In `src/menuConvert.dbd`, add a line for your new breakpoint table, using the breaktable name from the first line of `<bpname>.dbd`. Look at the existing breakpoint table entries in `menuConvert.dbd` for the proper format.
4. Two options depending on the monotonicity and base version
   - If the breakpoint table is monotonic or epics base is < 3.14.9, add `<bpname>.dbd` to `src/<appname>Support.dbd`:

   ```
   include <bpname>.dbd
   ```

   If Makefile is used instead of `<appname>Support.dbd`, add to `src/Makefile`:

   ```
   <appname>_DBD += <bpname>.dbd
   ```

   - If the breakpoint table is non-monotonic and epics base > 3.14.8, install `<bpname>.dbd` by itself in `src/Makefile`:

   ```
   DBD += <bpname>.dbd
   ```

5. Clean/build src.

6. Use the breaktable name in the record's LINR field. Make sure that the device support for the record supports conversion.
7. If the breakpoint table is non-monotonic and epics base > 3.14.8, change `st.cmd` to set the non-monotonic flag and load the breakpoint table:

```
dbBptNotMonotonic=1 (rtems, vxWorks)
```

or:

```
var dbBptNotMonotonic 1 (soft)
```

then:

```
dbLoadRecords("dbd/<bpname>.dbd")
```

1.15 EPICS Related Software

Tags: user developer all

This page attempts to list all EPICS-related source code and documentation outside of EPICS Base. If you find a link is incorrect or missing, please submit an issue or pull-request with a fix on the epics-docs repository. When submitting a pull-request, be sure to be familiar with our *documentation contribution guide*.

1.15.1 IOC Support Modules

These support modules are meant to be built into an IOC. See also the epics-modules project on github, there may be something there that has not yet been placed in this list.

Name	Source Code	Documentation
alive	github.com	github.io
Area Detector	github.com	github.io
asyn	github.com	github.io
autosave	github.com	github.io
busy	github.com	github.io
calc	github.com	github.io
camac	github.com	github.io
caputLog	github.com	github.com
caputRecorder	github.com	github.io
dac128V	github.com	github.io
Dante	github.com	github.io
delaygen	github.com	github.io
devlib2	github.com	github.io
dxp	github.com	github.io
dxpSITORO	github.com	github.io
ecmc	github.com	github.com
ecmccfg	github.com	github.io
ether_ip	github.com	github.com
fftw	github.com	github.com
gtest	github.com	github.io

continues on next page

Table 1 – continued from previous page

Name	Source Code	Documentation
gtr	github.com	epics.anl.gov
iocStats	github.com	slac.stanford.edu
ioczed	github.com	github.com
ip	github.com	epics.anl.gov
ip230A	github.com	millenia.cars.aps.anl.gov
ip330	github.com	github.io
ipac	github.com	epics.anl.gov
ipUnidig	github.com	github.io
LabJack	github.com	github.io
love	github.com	github.io
lua	github.com	github.io
mca	github.com	github.io
MCoreUtils	github.com	github.io
measComp	github.com	github.io
microEpsilon	github.com	github.com
modbus	github.com	github.io
motor	github.com	github.com
mrfioc2	github.com	sourceforge
nds3	github.com	github.io
opcua	github.com	github.com
optics	github.com	github.io
pcas	github.com	github.com
pmac	github.com	github.com
pyDevSup	github.com	github.io
quadEm	github.com	github.io
recsync	github.com	github.io
scaler	github.com	github.io
sequencer	github.com	sourceforge
snmp	groups.nscl.msu.edu	groups.nscl.msu.edu
softGlue	github.com	github.io
softGlueZynq	github.com	github.com
sscan	github.com	github.io
std	github.com	github.io
Stream Device	github.com	github.io
symb	github.com	github.com
SyringePump	github.com	github.com
tpmac	github.com	github.com
Transient Recorder	github.com	github.com
vac	github.com	github.io
vme	github.com	github.io
xspress3	github.com	github.io
xxx	github.com	github.io
Yokogawa_DAS	github.com	github.io

1.15.2 User Interface Tools

Graphical User Interface tools are an integral part of any EPICS installation. Being client tools, there is a variety of implementations using different programming languages and toolkits. Below is a list of the most commonly known ones.

Name	Description	Source Code	Documentation
CS-Studio (Phoebus)	Control System Studio (Java)	github.com	readthedocs.io
caQtDM	A display manager in the spirit of MEDM (C++, Qt)	github.com	github.io
EDM	Extensible Display Manager	github.com	controlssoftware.sns.ornl.gov
EPICS Qt	EPICS Qt framework	github.com	github.io
MEDM	Motif editor and display manager	github.com	epics.anl.gov
Probe	Motif channel monitoring program	github.com	epics.anl.gov
PyDM	A Python-based display manager	github.com	github.io
React Automation Studio	React-based display manager	github.com	github.com
Strip Tool	Strip-chart plotting tool	github.com	epics.anl.gov

1.15.3 Central Services

Name	Description	Source Code	Documentation
ALH	Alarm Handler (C, Motif)	github.com	epics.anl.gov
Archiver Applience (Java)	High performance, scalable process data archiver	github.com	github.io
BEAST (Java)	**DEPRECATED** Best Ever Alarm System Toolkit (within CS-Studio)	github.com	readthedocs.io
BEAUTY (Java)	**DEPRECATED** PV Archiver (Within CS-Studio)	github.com	sourceforge.net
BURT	Backup and restore tool	epics.anl.gov	epics.anl.gov
CASR	Host-based save/restore	github.com	epics.anl.gov
CA Gateway	PV gateway for channel access	github.com	epics.anl.gov
CA Watcher			
Channel Finder	Directory service for EPICS channels	github.com	github.io
Channel Watcher	Channel Watcher replaces the save part of EPICS save/restore	slac.stanford.ec	slac.stanford.edu
MASAR	Machine Snapshot, Archive and Restore	github.com	epics.anl.gov (pdf)
NameServer	Channel Access Name Server	github.com	epics.anl.gov
PVA Gateway	PV Access gateway	github.com	github.io

1.15.4 Language Bindings and Interfaces to Other Tools

C/C++

Name	Description	Source Code	Documentation
EPICS Base	Has CA/PVA implimentations in releases	github.com	epics-controls.org
EZCA	Eacy CA interface for C programs	github.com	epics.anl.gov
SCA	Simple Channel Access for C programs	als.lbl.gov	als.lbl.gov

Java

Name	Description	Source Code	Documentation
EPICS Core Java	Java implementation bundle	github.com	github.com
CA	Pure Java CA client	github.com	github.com
JCA	Java CA client API	github.io	javadoc.io
JCAE	Java CA Extensions library	github.com	github.com

LabView

Name	Description	Source Code	Documentation
CA Lab	CA client for Labview	github.com	github.com

Matlab

Name	Description	Source Code	Documentation
LabCA	Ca client library for Matlab	github.com	slac.stanford.edu
Matlab CA (MCA)	CA client library for Matlab	github.com	sns.ornl.gov

Perl

Name	Description	Source Code	Documentation
CAP5	CA for Perl 5	github.com	epics.anl.gov
PEZCA	A Perl binding to EZCA		

Python

Name	Description	Source Code	Documentation	Protocol
aioca	Asynchronous EPICS Channel Access client for asyncio and Python	github.com	github.io	CA
CaChannel	CaChannel	github.com	readthedocs.io	CA
caffi	Channel Access Foreign Function Interface	github.com	readthedocs.io	CA
caproto	Pure-python channel access	github.com	github.io	CA
cothread	Designed for cooperative threading (C, Python)	github.com	readthedocs.io	CA
pythonSoftIOC	Embed an EPICS IOC in a Python process	github.com	github.io	PVA/CA
p4p	Python wrapper around PVA client and server	github.com	github.io	PVA
pvapy	Python interface to pvAccess	github.com	epics.anl.gov	PVA/CA
pyepics	Python wrapper around libca	github.com	github.io	CA

Other

Name	Description	Source Code	Documentation
IDL	CA client libraries and scripts for IDL via EZCA	github.com	github.com
igor2epics	CA client library for WaveMetrics IGOR Pro	sourceforge.net	sourceforge.net
NetChannelAccess	CA libraries and Gateway in native C#	github.com	github.com
Node EPICS CA	CA client library for Node.js	npmjs.com	github.com
Node EPICS	**UNMAINTAINED** EPICS CA for node.js	github.com	github.com
SDDS	**ARCHIVED** The Self-Describing Data Sets analysis package	github.com	aps.anl.gov

1.15.5 IOC Database and Module Management Tools

Name	Description	Source Code	Documentation
dbVerbose	Verbose database filter	apics.anl.gov	apics.anl.gov
MSI	Macro substitution and include tool (in Base from 3.14)	github.com	epics.anl.gov
E3	ESS EPICS Environment Build System (Not limited to ESS)	gitlab.esss.lu.se	e3.pages.esss.lu.se
EPNix	Build, package, deploy IOCs and EPICS-related software using the Nix package manager	github.com	github.io
pyExpander	Python macro processing tool	sourceforge.io	sourceforge.io
pymsi	Python replacement for MSI		
SUMO	SUpport MOdule Manager	sourceforge.net	sourceforge.io
tdct	Visual hierarchical Database Configuration Tool, with configurable symbols		http://isacwserv.triumf.ca
VDCT	Visial Database Configuration Tool for EPICS databases	github.com	github.com

1.15.6 CA Server Interfaces and Applications

Name	Description	Source Code	Documentation
CAEX	Channel Access Examples	epics.anl.gov	epics.anl.gov
CAPod	Channel Access projects for Apple iOS devices	sourceforge.net	sourceforge.net
PCAS	Channel Access Server Library	github.com	epics.anl.gov
CaSnooper	Channel Access Search Request Diagnostic Tool	epics.anl.gov	epics.anl.gov
caxy	CA tunneling over ssh	github.com	github.io
JCAS	Pure Java CA server library	sourceforge,net	sourceforge.net
Kryten	Tool to run commands on PV changes	github.com	github.com
PCASpy	Python bindings for the CA server	github.com	readthedocs.io

1.15.7 Other Tools and Libraries

Name	Description	Source Code	Documentation
CA Shark	Wireshark dissector plugin for EPICS protocols	github.com	github.com
CMLOG	Common Message Logging System	jlab.org	jlab.org
EPICS GNU regex	The GNU regex library built with EPICS Makefiles	epics.anl.gov	epics.anl.gov
EdlBuild	Create EDM screens in Perl scripts	isacwserv.triumf.ca	isacwserv.triumf.ca
ParseCASW	CA beacon anomaly diagnostic tool	epics.anl.gov	epics.anl.gov
procServ	Process Server with Telnet Console	github.com	github.com
PViewer	Python 1D and 2D viewer	epics.anl.gov	epics.anl.gov
Wireshark CA	CA plug-in for Wireshark	wireshark.org	www-linac.kek.jp

1.15.8 (High Level) Application Packages

Name	Description	Source Code	Documentation
OpenXAL	Accelerator physics application framework (Java)	github.com	github.io
Matlab Middle Layer	Accelerator Physics Toolbox	github.com	github.com
React Automation Studio	Web-based EPICS interface	github.com	github.com

1.16 How To Port EPICS to a new OS/Architecture

`{tags} user `

This isn't a detailed list of tasks, but is intended to show the main stages needed to add a new build architecture to EPICS. If you make use of this and find there are hints you'd like to suggest, or steps missing please add them.

- Download a tarfile for the latest release of EPICS Base, or the snapshot from the R3.14 branch (not the trunk), and unpack it.
- If you're not already familiar with EPICS, at least skim chapter 4 of the IOC Application Developers Guide (hereafter known as the AppDevGuide; our build system is different to the usual "./configure && make" approach.
- Build your <base> on a linux-x86 or solaris-sparc system so you know what a fully built system actually looks and acts like. You can build multiple architectures simultaneously in the same tree, which makes for easier comparisons. On linux the build instructions should be as simple as

```
export EPICS_HOST_ARCH=linux-x86
cd <base>
make
```

- On the new system system, setenv EPICS_HOST_ARCH to the name for your new architecture, which usually takes the form <osname>-<cpufamily>, for example solaris-sparc, linux-x86, windows-x86
- In the <base>/configure/os directory, create these files by copying and editing the files from an existing architecture:

```
CONFIG.Common.<arch>
CONFIG.<arch>.Common
CONFIG_SITE.<arch>.Common
```

- I would suggest looking at the darwin-ppc or linux-x86 versions to start with; for a Unix-like OS you should be able to make use of the UnixCommon and/or GnuCommon files to provide most of the definitions and rules.
- If you have to cross-compile then there's more work you have to do and these instructions are probably not sufficient to get you there.

1.17 PV Access repositories overview

1.18 EPICS V4 Normative Types

Tags: developer advanced

1.18.1 EPICS V4 Normative Types, Editors Draft, 16-Mar-2015

Editors:
Greg White, SLAC Bob Dalesio, BNL Mark Rivers, APS (Invited Expert) Marty Kraimer, BNL David Hickin, Diamond Light Source

---

1.18.2 Abstract

This document defines a set of standard high-level data types to aid interoperability of peers at the application level of an EPICS V4 network.

The abstract type definition and function of each such standard type is described. For instance, one such type defined here, named "NTTable", defines a structure for expressing (in pvData) and communicating (using pvAccess) a table of numeric or string data.

The data types described here are approximately equivalent to EPICS V3 Database Request types (commonly known as "DBR" types), although Normative Types extend the concept to structured data and operate at a higher level in a complex control system, or data exchange, than DBR types. Also, Normative Types may be used purely for data exchange though the dynamic data exchange interfaces offered by EPICS pvAccess and pvData modules, such as pvDatabase or pvAccess RPC servers.

For more information about EPICS, please refer to the home page of the Experimental Physics and Industrial Control System.

1.18.3 Status of this Document

This revision of the Normative Types document is a minor modification to the 16 Mar 2015 version. This revision adds minor clarifications to the description of NTTable.

The 16 Mar 2015 version updates the definitions of time_t, control_t, display_t, and alarmLimit_t and changes the order of optional fields in a number of Normative Types. It replaces NTImage with NTNDArray, adds NTAttribute, NTMultiChannel, NTUnion and NTScalarMultiChannel and removes NTVariantArray and a number of types proposed in earlier drafts.

This version contains a number of types which use pvData unions.

It describes the new conventions for Normative Type IDs including versioning and namespaces. Type IDs for Normative Type structure fields are given.

The linguistic conventions used in the document have been overhauled.

See *Appendix A* for items that may be added to future revisions of this specification.

This version is an Editors Draft towards the First Public Working Draft. The First Public Working Draft will be intended for the EPICS community to review and comment. Resulting comments will drive subsequent revisions of the Normative Types specification and the EPICS V4 Working Group's reference implementations of software that helps create, populate and exchange Normative Type pvData.

Comments are welcome, though bear in mind this is a pre-public release version.

The terms MUST, MUST NOT, SHOULD, SHOULD NOT, REQUIRED, and MAY when highlighted (through style sheets, and in uppercase in the source) are used in accordance with RFC 2119 [RFC2119]. The term NOT REQUIRED (not defined in RFC 2119) indicates exemption.

1.18.4 Table of Contents

1.18.5 Introduction

The Normative Types described in this document are a set of software designs for high-level composite data types suitable for the application-level data exchange between EPICS network endpoints using the pvAccess protocol. In particular, they are intended for use in online scientific data services. The intention is that where the endpoints in an EPICS network use only Normative Types, each peer in the network should be able to understand all the data transmitted to it, at least syntactically, and be able to take processing steps appropriate to that data.

We call these types the Normative Types, to emphasize their role as the prescriptions of abstract data structures, whose role and intended semantics are described in this document, as opposed to implemented software; and that conformance to these semantics is a necessary condition for interoperability of using systems.

The EPICS (7) module pvData *bib:pvdata* supplies a typing mechanism and object management API for efficiently defining, creating, accessing and updating memory resident structured data. EPICS module pvAccess *bib:pvaccess* supports the efficient exchange of pvData defined data between EPICS V4 network peers. The EPICS V4 Normative Types specification defines some general purpose data types that build on pvData. These are designed to be generally applicable to the process control, and the software applications level, of scientific instruments.

A simple example of a Normative Type described in this document is the one for exchanging any single scalar value, such as one floating point number, one integer or one string. That Normative Type is named "NTScalar". When a client receives a pvData datum which identifies itself as being of type NTScalar, the client will know to expect that the structure which carries the NTScalar will include the scalar value in question (along with its type), and that value may be accompanied by up to 5 additional fields: a description of the quality in question, a timestamp, an indication of alarm severity, fields that help in how to display the value, and data about its operating limits. See the example below.

An example of a simple Normative Type is the NTScalar:

```
NTScalar :=

structure
    scalar_t    value
    string      descriptor  :opt
    alarm_t     alarm       :opt
    time_t      timeStamp   :opt
    display_t   display     :opt
    control_t   control     :opt
```

A more complex example: If a client receives a pvData datum which identifies itself as being of type NTTable, this document specifies that it should expect the datum to contain 0 or more arrays of potentially different types. The description of NTTable in this document will say that the client should interpret the arrays as the columns of a table, and should render such a datum appropriately as a table, with row elements being taken from the same numbered elements of each array.

```
NTTable :=

structure
  string[]    labels           // The field names of each field in value
  structure   value
     {scalar_t[] colname}0+  // 0 or more scalar array type
                              // instances, the column values.
  string      descriptor    : opt
  alarm_t     alarm         : opt
  time_t      timeStamp     : opt
```

1.18.6 Description of Normative Types

All the EPICS V4 Normative Types are defined as particular structure instance definitions of a pvData structure. This is true even of the Normative Types describing simple values like a single int, since all Normative Types optionally include descriptor, alarm and timestamp. The fields of any given Ntype datum instance can be ascertained at runtime using the pvData Field introspection interface *bib:pvdata*.

See the *Normative Type instance self-identification* section below for more on how to examine a given pvData instance to see which fields it includes. That section also includes how to mark a pvData instance as a Normative Type, and how to look for that mark.

Definition: Normative Type

The Normative Types definitions in this document each have the following general form:

1. They are defined as structures, composed of fields.
2. They usually have one primary field called "value", which encodes the most important data of the type.
3. They are composed of required fields, and optional fields. The required fields come first, the optional fields follow.
4. The order of fields matters. Although the Normative Types pvData binding allows for access though an introspection API, senders must encode the fields in the order described in this document.

Linguistic conventions used in this document

A Normative Type can be used both for sending data from client to service and from service to client. In this document we refer generally to an *agent*, being either a client or a server. If the agent is specifically at the user's end, we call it the *user agent*. *Client* and *server* refer to the directionality of the transaction, server being the agent that is doing the sending.

The word "Ntype" is used as a short form of "Normative Type".

The Normative Type data descriptions are given with the syntactic conventions and grammar given below. The types are described in a BNF-like syntax in order to add clear distinctions between symbol types, particularly terminality, recurrence, which names a user is expected to add and which are predefined. This syntax is essentially Extended Backus-Naur Form (EBNF), with some slight modifications to preserve the order of terms and the rules for line ends and indentation.

The syntactic conventions are as follows.

First, the conventions for terminal and non-terminal types are:

- *italics* - a non-terminal. These are used to stand for a choice of pvData type, or named sequence of fields, or for a specific structure or union, and hence non-terminal.
- `plaintext` - terminals. These will be either a pvData Meta Language keyword or a label. The Meta language keywords consist of `structure`, `union`, `any`, the scalar type keywords (`boolean`, `byte`, `short`, `int`, `long`, `double`, `ubyte`, `ushort`, `uint`, `ulong`, `float`, `double` and `string`) and the corresponding arrays `structure[]`, `union[]`, `any[]`, and scalar arrays (e.g. `int[]`, `double[]`).
- <name> - A user-provided label name.A programmer using the Normative Type will choose what goes in the <>.

So, for example, *scalar_t* is non-terminal as it stands for a choice of pvData type and *time_t* is non-terminal because it stands for a particular structure. On the other hand, in the definition of *time_t*, `long` and `secondsPastEpoch` are a keyword and a label respectively, and so are terminal, and the columns of *NTTable*, <colname>, are user-provided labels.

In this section <> will also be used for describing patterns of definitions or meta rules such as production rules of the grammar to indicate a choice of terminal or non-terminal terms in the pattern or rule.

The EBNF-like syntax for definitions is used. A description consists of 3 terms - a left-hand side (LHS), a right-hand side (RHS), and the symbol ":=" separating them, which is to be interpreted as "LHS is defined as RHS". The LHS will be the non-terminal being defined. The RHS will be a sequence of terminal or non-terminal terms.

Note that in the definitions below line-ends (EOLs) are not explicitly specified. They are implied except when multiple lines are used to specify alternatives separated by |, where only the final EOL is implied.

The following EBNF symbols are also used:

- | - used to separate alternative items; one item is chosen from this list of alternatives.
- [] - optional items are enclosed between square brackets [and]; the item can either be included or discarded. Note, optional fields of structures are marked as such by the use of `:opt` instead of square brackets.
- { } - a sequence of occurrences of the item or items in the braces. The number of occurrences follows. 0+ means 0 or more. 1+ means 1 or more.

The following production rules are employed:

1. Replace a non-terminal by its definition, except where the non-terminal defines a structure or union and is followed by a field name. (The modified rule for non-terminal structures and unions is described below.)
2. Choose an alternative for items separated by |.
3. Choose a user supplied label for items between angle brackets (< and >).
4. Include or discard items between square brackets ([and]). Note this excludes a pair of square brackets ([]) used to signify an array.
5. Include or discard fields marked `:opt`.
6. For items between braces ({ and }) replace with an appropriate number of occurrences of the item. For a sequence of pvData fields a line-end (EOL) is implied after each one.

In the case of structure and union fields, to preserve the order of terms in the pvData Meta language, as well as obtaining appropriate indentation, the usual EBNF rule of replacing a non-terminal by its definition requires the following modification:

Suppose a non-terminal term has a definition of the form

```
<non-terminal>:=

structure
    fieldList
```

where:

<non-terminal>
: The non-terminal term being defined.

<fieldName>

: A choice of terminal or non-terminal terms describing a list of 0 or more pvData fields.

Then for a label (a field name), <fieldName>, the terms

```
<non-terminal> <fieldName>
```

are replaced by

```
structure <fieldName>
    <fieldList>
```

The result of the any substitution is suitably indented to preserve the logic of the pvData meta language.

Thus the structure derived from the definition of *NTEnum* below, with all optional fields present, is

```
structure
    structure  value
        int      index
        string[] choices
    string     descriptor
    structure  timeStamp
        long     secondsPastEpoch
        int      nanoseconds
        int      userTag
    structure   alarm
        int      severity
        int      status
        string   message
```

The same rule also applies with `union` in place of `structure`.

The grammar for a Normative Type definition follows the pattern below. That is, a Normative Type is defined as a structure composed of fields. A field may be optional, and may be described along with a comment:

```
<NormativeType>:=

structure
   { <pvDataField> [:opt] [// <commentText>] }1+
```

where:

<*NormativeType*>
: The name of the Normative Type being defined.

<pvDataField>
: A choice of terms defining a pvData field

:opt
: Indicates that the preceding field is optional in the Normative Type.

// <commentText>
: A field production element may be followed by a comment.

In most cases a Normative Type definition will be of the form

```
<NTname>:=

structure
   { ntfieldChoice fieldName [:opt] [// commentText] }1+
```

where:

<ntFieldChoice>
: Terminal or non-terminal terms, possibly separated by |, from the valid *Normative Type Fields* as defined below.

<fieldName>
: The identifier of the field. Usually a terminal label.

For example, a definition meeting this pattern would be

```
NTExample :=

structure
    enum_t | scalar_t   value
    int                 N                   // this field has a comment
    string              descriptor  :opt
    alarm_t             alarm       :opt
    time_t              timeStamp   :opt
```

1.18.7 Normative Type Fields

This section defines the fields that may appear in a Normative Type's definition.

Each field of a Normative Type will typically be one of the following:

```
ntfield :=

  scalar_t        // a simple numerical, boolean, or string value
| scalar_t[]      // an array of simple values
| enum_t          // an enumeration
| enum_t[]        // an array of enumerations
| time_t          // a point in time, used for timestamps
| time_t[]        // an array of points in time
| alarm_t         // a summary diagnostic of a control system event
| alarm_t[]       // an array of summary diagnostics
| alarmLimit_t    // value thresholds for a control system diagnostic report
| alarmLimit_t[]  // an array of threshold values
| display_t       // metadata of displayed data
| display_t[]     // an array of display metadata
| control_t       // control setpoint range boundaries
| control_t[]     // an array of control setpoint range boundaries
| any             // a variant union type
| any[]           // an array of variant unions fields
| ntunion_t       // a regular union storing ntfields only
| ntunion_t[]     // a regular union array storing ntfields only
| union_t         // any regular union
| union_t[]       // any regular union array
| anyunion_t      // any variant or regular union
| anyunion_t[]    // any variant or regular union array
```

although some examples may have fields of other types.

Simple Normative Type fields - scalar and scalar array types

Note that of all the Normative Type fields only *scalar_t* and *scalar_t[]* are of simple type, that is, having a single scalar or scalar array value of a fixed type. All the others are represented by a complex type, i.e. a structure or union or arrays of structures or unions (see *Structured Normative Type fields* and *Union Normative Type fields* below).

scalar_t

The field is a scalar value. Scalar fields would be implemented with pvData field Type "scalar":

```
scalar_t :=

   boolean  // true or false
|  byte     // 8 bit signed integer
|  ubyte    // 8 bit unsigned integer
|  short    // 16 bit signed integer
|  ushort   // 16 bit unsigned integer
|  int      // 32 bit signed integer
|  uint     // 32 bit unsigned integer
|  long     // 64 bit signed integer
|  ulong    // 64 bit unsigned integer
|  float    // single precision IEEE 754
|  double   // double precision IEEE 754
|  string   // UTF-8 *
```

scalar_t[]

The field is an array of scalars. Scalar array fields would be implemented with a pvData field of type "scalarArray":

```
scalar_t[] :=

   boolean[]  // array of true or false
|  byte[]     // array of 8 bit signed integer
|  ubyte[]    // array of 8 bit unsigned integer
|  short[]    // array of 16 bit signed integer
|  ushort[]   // array of 16 bit unsigned integer
|  int[]      // array of 32 bit signed integer
|  uint[]     // array of 32 bit unsigned integer
|  long[]     // array of 64 bit signed integer
|  ulong[]    // array of 64 bit unsigned integer
|  float[]    // array of single precision IEEE 754
|  double[]   // array of double precision IEEE 754
|  string[]   // array of UTF-8 *
```

Structured Normative Type fields

This subsection defines those fields of a Normative Type structure definition that are themselves structures or arrays of structures.

The structured Normative Type fields would be implemented with type pvData field type "structure" or "structureArray".

enum_t

An *enum_t* describes an enumeration. The field is a structure describing a value drawn from a given set of valid values also given. It is implemented as a pvData Field of type "structure" of type ID "enum_t" with the following form:

```
enum_t :=

structure
    int index
    string[] choices
```

where:

index
The index of the current value of the enumeration in the array choices below.

choices
An array of strings specifying the set of labels for the valid values of the enumeration.

enum_t[]

An *enum_t[]* describes an array of enumerations. The field is an array of structures each describing a value drawn from a given set of valid values also given in each. It is implemented as a pvData field of type "structureArray", each element of which is a structure of the form *enum_t* above.

time_t

A *time_t]* describes a defined point in time. The field is a structure describing a time relative to midnight on January 1st, 1970 UTC. It is implemented as a pvData field of type "structure" of type ID "time_t" and with the following form:

```
time_t :=

structure
    long secondsPastEpoch
    int  nanoseconds
    int  userTag
```

where:

secondsPastEpoch
Seconds since Jan 1, 1970 00:00:00 UTC.

nanoseconds
Nanoseconds relative to the `secondsPastEpoch` field.

userTag
An integer value whose interpretation is deliberately undefined and therefore MAY be used by EPICS V4 agents in a user defined way.

Interpretation: The point in time being identified by a *time_t*, is given by Jan 1, 1970 00:00:00 UTC plus some nanoseconds given by its `secondsPastEpoch` times 10^9 plus its `nanoseconds`.

time_t[]

A *time_t[]* describes an array of points in time. The field is an array of structures each describing a time relative to January 1st, 1970 UTC. It is implemented as a pvData field of type "structureArray", each element of which is a structure of the form *time_t* above.

alarm_t

An *alarm_t* describes a diagnostic of the value of a control system process variable. It indicates essentially whether the associated value is good or bad, and whether agent systems should alert people to the status of the process.

Processes in EPICS V3 and V4 IOCs include extensive support for evaluating alarm conditions. The definition of the fields in an `alarm` are given in *bib:epicsrecref*. The field is a structure describing an alarm. It is implemented as a pvData field of type "structure" of type ID "alarm_t" with the following form:

```
alarm_t :=

structure
    int severity
    int status
    string message
```

where:

severity
: severity is defined as an int (not an *enum_t*), but MUST be functionally interpreted as the enumeration {noAlarm, minorAlarm, majorAlarm, invalidAlarm, undefinedAlarm } indexed from noAlarm=0 *bib:epicsrecref*.

status
: status is defined as an int (not an *enum_t*), but MUST be functionally interpreted as the enumeration {noStatus, deviceStatus, driverStatus, recordStatus, dbStatus, confStatus, undefinedStatus, clientStatus } indexed from noStatus=0 *bib:epicsrecref*.

message
: A message string.

Interpretation MUST be as with V3 IOC record processing, as described in the EPICS Reference Manual *bib:epicsrecref*.

alarm_t[]

An *alarm_t[]* is an array of alarm conditions. The field is an array of structures each describing an alarm condition. It is implemented as a pvData field of type "structureArray", each element of which is a structure of the form *alarm_t* above.

alarmLimit_t

An *alarmLimit_t* is a structure that gives the numeric intervals to be used for the high and low limit ranges of an associated value field. The specific value to which the alarmLimit refers, is not specified in the alarmLimit structure. It is usually a value field of type double that appears in the same structure as the alarmLimit. *alarmLimit_t* is implemented as a pvData field of type "structure" of type ID "alarmLimit_t" with the following form:

```
alarmLimit_t :=

structure
    boolean active
    double lowAlarmLimit
    double lowWarningLimit
    double highWarningLimit
    double highAlarmLimit
    int lowAlarmSeverity
    int lowWarningSeverity
    int highWarningSeverity
    int highAlarmSeverity
    double hysteresis
```

where:

active
: Is alarming active? If no then alarms are not raised. If yes then the associated value is checked for alarm conditions.

lowAlarmLimit
: If the value is <= lowAlarmLimit then the severity is lowAlarmSeverity.

lowWarningLimit
: If the value is > lowAlarmLimit and <= lowWarningLimit then the severity is lowWarningSeverity.

highWarningLimit
: If the value is >= highWarningLimit and < highAlarmLimit then the severity is highWarningLimit.

highAlarmLimit
: If the value is >= highAlarmLimit then the severity is highAlarmSeverity.

lowAlarmSeverity
: Severity for value that satisfies lowAlarmLimit.

lowWarningSeverity
: Severity for value that satisfies lowWarningLimit.

highWarningSeverity
: Severity for value that satisfies highWarningLimit.

highAlarmSeverity
: Severity for value that satisfies highAlarmLimit.

hysteresis
: When a value enters an alarm limit this is how much it must change before is it put into a lower severity state. This prevents alarm chatter.

Code that checks for alarms should use code similar to the following:

```
boolean active = pvActive.get();
if(!active) return;
```

(continues on next page)

(continued from previous page)

```
double  val = pvValue.get();
int severity = pvHighAlarmSeverity.get();
double level = pvHighAlarmLimit.get();
if(severity>0 && (val>=level)) {
    raiseAlarm(level,val,severity,"highAlarm");
    return;
}
severity = pvLowAlarmSeverity.get();
level = pvLowAlarmLimit.get();
if(severity>0 && (val<=level)) {
    raiseAlarm(level,val,severity,"lowAlarm");
    return;
}
severity = pvHighWarningSeverity.get();
level = pvHighWarningLimit.get();
if(severity>0 && (val>=level)) {
    raiseAlarm(level,val,severity,"highWarning");
    return;
}
severity = pvLowWarningSeverity.get();
level = pvLowWarningLimit.get();
if(severity>0 && (val<=level)) {
    raiseAlarm(level,val,severity,"lowWarning");
    return;
}
raiseAlarm(0,val,0,"");
```

NOTE: The current pvData implementations have a structure named **valueAlarm_t** instead of **alarmLimit_t**. *valueAlarm_t* is similar to *alarmLimit_t*, except that the former's alarm limit fields (`lowAlarmLimit`, `lowWarningLimit`, `highWarningLimit` and `highAlarmLimit`) can be any integer or floating point scalar type (the same type for all the limit fields in each case), rather than only double. There is also a separate form for alarm limits for boolean values. *alarmLimit_t* is identical to the *valueAlarm_t* for type double, except that the type ID of *valueAlarm_t* is "valueAlarm_t"). Normative types only defines alarmLimit since this is what clients like plot tools use.

alarmLimit_t[]

An *alarmLimit_t[]* is an array of alarm limit conditions. The field is an array of structures each describing an alarm limit. It is implemented as a pvData field of type "structureArray", each element of which is a structure of the form *alarmLimit_t* above.

display_t

A *display_t* is a structure that describes some typical attributes of a numerical value that are of interest when displaying the value on a computer screen or similar medium. The `units` field SHOULD contain a string representation of the physical units for the value, if any. The `description` field SHOULD contain a short (one-line) description of what the value represents, such as can be used as a label in a display. The fields `limitLow` and `limitHigh` represent the range in between which the value should be presented as adjustable.

The field is a structure describing a *display_t*. It is implemented as a pvData field of type "structure" of type ID "display_t" with the following form:

```
display_t :=

structure
    double limitLow
    double limitHigh
    string description
    string units
    int precision
    enum_t form(3)
        int index
        string[] choices ["Default", "String", "Binary", "Decimal", "Hex", "Exponential",
↪ "Engineering"]
```

where:

limitLow
The lower bound of range within which the value must be set, to be presented to a user.

limitHigh
The upper bound of range within which the value must be set, to be presented to a user.

description
A textual summary of the variable that the value quantifies.

precision
Number of decimal points that are displayed when formatting a floating point number. This corresponds to the PREC field in EPICS database records with floating point values (e.g., ai, ao, calc, calcout record types.)

form
An enumeration to specify formatting a value to be displayed. By default, a floating point number is formatted with the number of decimal points defined in the precision field. Formatting of an EPICS database record value can be configured by including eg. info(Q:form, "Hex") in record definition.

units
The units for the value field.

Where an *display_t* structure instance is present in a Normative Type structure, it MUST be interpreted as referring to that Normative Type's field named "value". Therefore it is only used in Normative Types that have a single numeric "value" field.

display_t[]

A *display_t[]* is an array of *display_t*. The field is an array of structures each describing the display media oriented metadata of some corresponding process variable value, as described by *display_t* above. It is implemented as a pvData field of type "structureArray", each element of which is a structure of the form *display_t* above.

control_t

A *control_t* is a structure that describes a range, given by the interval (limitLow,limitHigh), within which it is expected some control software or hardware shall bind the control PV to which this Normative Type instance's value field refers as well as a minimum step change of the control PV.

The field is a structure describing a *control_t*. It is implemented as a pvData field of type "structure" of type ID "control_t" with the following form:

```
control_t :=

structure
    double limitLow
    double limitHigh
    double minStep
```

where:

lowLimit
: The control low limit for the value field.

highLimit
: The control high limit for the value field.

minStep
: The minimum step change for the value field.

control_t[]

A *control_t[]* is an array of *control_t*. The field is an array of structures each describing the setpoint range interval of some process variable. It is implemented as a pvData field of type "structureArray", each element of which is a structure of the form *control_t* above.

Union Normative Type fields

This subsection defines those fields of a Normative Type structure definition that are unions or arrays of unions.

The union NormativeType fields are implemented with pvData fields of type "union" or "unionArray".

The union Normative Type fields consist of the variant union `any` and variant union array `any\[\]` as well as a number of non-terminal terms:

any

This is a field which is a variant union and is implemented using the pvData field type "union".

any[]

This is a field that is an array of any, implemented using the pvData field type "unionArray".

ntunion_t

ntunion_t stands for any regular union of ntfields and is implemented using the pvData field type "union":

```
ntunion_t :=

union
    {ntfield  field-name}1+  // 1 or more ntfields.
```

ntunion_t[]

An *ntunion_t[]* stands for an array of unions, where the union is any regular union of 1 or more ntfields. It is implemented as a pvData field of type "unionArray" each element of which is a union (the same one in each case) of the form *ntunion_t* above.

union_t

union_t stands for any regular union of pvData fields and is implemented using the pvData field of type "union":

```
union_t :=

union
    {pvDataField}1+ // 1 or more pvData fields.
```

where:

pvDataField
: Stands for any pvData field.

union_t[]

A *union_t[]* stands for an array of unions, where the union is any regular union of 1 or more pvData fields. It is implemented as a pvData field of type "unionArray" each element of which is a union (the same one in each case) of the form *union_t* above.

anyunion_t

anyunion_t stands for a variant union or any regular union of pvData fields and is implemented using the pvData field type "union":

```
anyunion_t:=

any | union_t
```

anyunion_t[]

An *anyunion_t[]* stands for a variant union array or a regular union array of any type an array of unions, where the union is any regular union of 1 or more pvData fields. It is implemented as a pvData field of type "unionArray" each element of which is a union (the same one in each case) of the form *anyunion_t* above:

```
anyunion_t[]:=

any[] | union_t[]
```

1.18.8 Normative Type Metadata

Metadata are included in runtime instances of Normative Types. The metadata includes to which Normative Type the structure instance conforms, version information, and other data to aid efficient processing, diagnostics and displays.

Normative Type instance self-identification

Normative Type instance data MUST identify themselves as such by including an identifying string. That is the Normative Type Identifier, or "Ntype Identifier" string for short. In the pvData binding of Normative Types, this string is carried in the type ID, added automatically to every pvData structure.

A Normative Type Identifier MUST be considered to be "case sensitive."

The namespace Name of EPICS Normative Types (which is used as the prefix for their pvData type ID), is the following:

```
epics:nt
```

The normative list of the Normative Type Identifiers corresponding to *this draft* of the EPICS V4 Normative Types specification document (this document), is given in *Appendix B*

As an example, one of the simplest Normative Types is *NTScalar*. It has formal Type Name "NTScalar". Therefore, the Normative Type Identifier for an NTScalar, is presently epics:nt/NTScalar:1.0.

At present it is envisaged that the same namespace value shall be used for all versions of this document prior to Recommendation, including all Public Working Drafts of this document and those marked Last Call or similar.

pvAccess binding type identification

In the EPICS v4 pvData/pvAccess binding, the structure identification string (ID) of pvData structures is used to communicate the Normative Type of the datum carried by the pvData structure. Every pvData datum which is intended to conform to a Normative Type, MUST identify the Normative Type to which it conforms through its type ID. Its ID MUST have the value of its Normative Type Identifier. For instance, a pvData structure conforming to NTScalar, must have ID equal to "epics:nt/NTScalar:1.0". Every EPICS V4 agent which is encoding or decoding pvData data that is described by Normative Types, SHOULD examine the ID of such data, to establish the Normative Type to which each datum conforms.

Example pvAccess/pvData binding

Recall that in the pvData system, data variables are constructed in two equally important parts; the introspection interface, in which data types are defined, and the data interface, in which instance variables are created and populated. The introspection interface can be used to examine an existing instance, to see what fields it possesses. Getting and setting values, is done through the data interface. As a programmer, you have to define both parts, the introspection interface of your type, and its data interface. Both the data and the introspection interfaces are exchanged by pvAccess. That is, when a sender constructs a data type, such as one conforming to an Normative Type, plus an instance of that type, and it sends the instance to a receiver, the receiver can check that the instance indeed contains the member fields it should find for that type, using the type's introspection interface.

The following Java code snippets give an example of the use of a pvData structure of Normative Type *NTScalar*, as defined below. in this example we show code as may be included in a trivial "multiplier" service, and a client of the multiplier service.

Sender

The sender typically first creates an introspection definition, using the pvData introspection interfaces (Field, Structure etc.). It then creates an instance of the type and populates it with the pvData data interfaces (PVField, PVStructure etc.).

Example of creating the introspection interface of an NTScalar, as may be done on a server that will be returning one. In this example, only one of the optional fields of NTScalar, named "descriptor" is included, along with the required field named "value".

```
// Create the data type definition, using the pvData introspection interface (Structure␣
↪etc.).
FieldCreate fieldCreate = FieldFactory.getFieldCreate();
Structure resultStructure = fieldCreate.createStructure( "epics:nt/NTScalar:1.0",
    new String[] { "value", "descriptor" },
    new Field[] { fieldCreate.createScalar(ScalarType.pvDouble),
                  fieldCreate.createScalar(ScalarType.pvString) } );
```

Subsequently, the sender would create an instance of the type, and populate it.

Example of creating an instance and data interface of an NTScalar, as may be done on a data server, and populating it.

```
// If a and b were arguments to this service, the following creates an instance of
// a resultStructure, which conforms to the NTScalar Normative Type definition,
// and populates it. It would then return this PVStructure instance.
PVStructure result = PVDataFactory.getPVDataCreate().createPVStructure(resultStructure);
result.getDoubleField("value").put(a * b);
result.getStringField("descriptor").put("The product of arguments a and b");
```

The PVStructure instance, in the example called "result" would be returned to the receiver.

Receiver

Having in some way done a pvAccess get, the receiver could simply extract the primary value:

```
PVStructure result = easyPVA.createChannel("multiplierService").createRPC().
↪request(request);
double product = result.getDoubleField("value").get();
```

A well written receiver would check that the introspection interface (Structure etc.) says that the received instance is indeed of the type it expects. It may extract the data fields individually, checking their type. Importantly, it can also see which optional fields it received, before attempting to access them. Here is a more complete receiver example for the NTScalar sent above. This code might be in the client side of the Multiplier service.

Example of a receiver of an NTScalar. The example checks that the returned pvData datum was an instance of an NTScalar, extracts the required value field, and then, if it's present, extracts the optional "descriptor" field.

```
// Call the multiplier service sending the request in a structure
PVStructure result = easyPVA.createChannel("multiplierService").createRPC().
↪request(request);

// Examine the returned structure via its introspection interface, to check whether its
// identifier says that it is a Normative Type, and the type we expected.
if (!result.getStructure().getID().equals("epics:nt/NTScalar:1.0"))
{
    System.err.println("Unexpected data identifier returned from multiplierService: " +
       "Expected Normative Type ID epics:nt/NTScalar:1.0, but got "
       + result.getStructure().getID());
    System.exit(-1);
}

// Get and print the required value member field as a Double.
System.out.println( "value = " + result.getDoubleField("value").get());

// See if there was also the descriptor subField, and if so, get it and print it.
PVString descriptorpv = (PVString)result.getSubField("descriptor");
if ( descriptorpv != null)
   System.out.println( "descriptor = " + descriptorpv.get());

// Or just print everything we got:
System.out.println("\nWhole result structure toString =\n" + result);
```

Future of type identification

In future drafts of this specification, a pattern to create extensions to the EPICS V4 Normative Types may be presented. It may be based on a formalized link to the XML namespace and XML Schema system, whereby the namespace part of the Normative Type Identifier of a datum whose type is an extension of one of these Normative Types, is replaced by another namespace that extends this one through an XML Schema out of band. In that case, the type name part would identify a type in that other namespace, though it may extend a type in this namespace.

Standard optional metadata fields

All of the Normative Types defined below, optionally include a descriptor, alarm and timestamp. There is no required interpretation of these fields, and therefore their meaning is not further described in the Normative Type definitions. Additionally, Normative Types may have other optional fields, as defined individually below.

Optional descriptor field

An object of Normative Type may optionally include a field named "descriptor" and of type string, to be used to give identity, name, or sense information. For instance, it may be valued with the name of a device associated with control data, or the run number of a table of model data.

```
string descriptor  :opt     // Contextual information
```

Optional alarm field

An object of Normative Type may optionally include an alarm field.

```
alarm_t alarm      :opt     // Control system event summary
```

Optional timeStamp field

An object of Normative Type may optionally include a timeStamp field.

```
time_t timeStamp   :opt     // Event time
```

1.18.9 General Normative Types

The General Normative Types are for encapsulating data of any kind of application or use case. Compare to *Specific Normative Types*, defined later in this document, which are oriented to particular use cases.

NTScalar

NTScalar is the EPICS V4 Normative Type that describes a single scalar value plus metadata:

```
NTScalar :=

structure
    scalar_t    value
    string      descriptor  :opt
    alarm_t     alarm       :opt
    time_t      timeStamp   :opt
    display_t   display     :opt
    control_t   control     :opt
```

where:

value
: The primary data carried by the NTScalar object. The field must be named "value" and can be of any simple scalar type as defined above.

NTScalarArray

NTScalarArray is the EPICS V4 Normative Type that describes an array of values, plus metadata. All the elements of the array of the same scalar type.

```
NTScalarArray :=

structure
    scalar_t[]  value
    string      descriptor  :opt
    alarm_t     alarm       :opt
    time_t      timeStamp   :opt
    display_t   display     :opt
    control_t   control     :opt
```

where:

value
: The primary data carried by the NTScalarArray object. The field must be named "value" and can be of any scalar array type as defined above.

NTEnum

NTEnum is an EPICS V4 Normative Type that describes an enumeration (a closed set of possible values each described by an n-tuple).

```
NTEnum :=

structure
    enum_t      value
    string      descriptor  :opt
    alarm_t     alarm       :opt
    time_t      timeStamp   :opt
```

where:

value
: The primary data carried by the NTEnum object. The field must be named "value" and must be an enumeration as defined above.

NTMatrix

NTMatrix is an EPICS V4 Normative Type used to define a matrix, specifically a 2-dimensional array of real numbers.

```
NTMatrix :=

structure
    double[]    value
    int[2]      dim         :opt
    string      descriptor  :opt
    alarm_t     alarm       :opt
    time_t      timeStamp   :opt
    display_t   display     :opt
```

where:

value
: The numerical data comprising the matrix. The value is given as a single array of doubles. When `value` holds the data of a matrix, rather than a vector, then the data MUST be laid out in "row major order"; that is, all the elements of the first row, then all the elements of the second row, and so on. For instance, where NTMatrix represented a 6x6 matrix, element (1,2) of the matrix would be in the 2nd element of `value`, and element (3,4) would be in the 16th element.

dim
: `dim` indicates the dimensions of the matrix. If `dim` is not present, `value` MUST be interpreted as a vector, of length equal to the number of elements of `value`. If `dim` is present, then it must have 1 or 2 elements; its one element value or both elements values MUST be > 0, and the number of elements in `value` MUST be equal to the product of the elements of `dim`. If `dim` is present and contains a single element, then the NTMatrix MUST be interpreted as describing a vector. A `dim]` of 2 elements describes a matrix, where the first element of `dim` gives the number of rows, and the second element of `dim` gives the number columns. If `dim` is present and contains 2 elements, of which the first is unity, and the second is not (therefore is >1) then the NTMatrix MUST be interpreted as describing a row vector. If `dim` is present as contains 2 elements, of which the second is unity, and the first is not (therefore is >1) then the NTMatrix MUST be interpreted as describing a column vector.

User agents that print or otherwise render an NTMatrix SHOULD print row vector, column vector, and non-vector matrices appropriately.

NTURI

NTURI is the EPICS V4 Normative Type that describes a Uniform Resource Identifier (URI) *bib:uri*. Specifically, NTURI carries the four parts of a "Generic URI", as described in *bib:uri* as the subset of URI that share a common syntax for representing hierarchical relationships within the namespace. As such, NTURI is intended to be able to encode any generic URI scheme's data. However, NTURI's primary purpose in the context of EPICS, is to offer a well formed and standard compliant way that EPICS agents can make a request for an identified resource from a channel, especially an EPICS V4 RPC channel. See ChannelRPC.

The "pva" scheme is introduced here for EPICS V4 interactions. The pva scheme implies but does not require use of the pvAccess protocol. A scheme description for Channel Access (implying the ca protocol) will be added later. What follows is a description of the syntax and semantics for the pva scheme.

```
NTURI  :=

structure
    string scheme
    string authority   : opt
    string path
    structure query    : opt
        {string | double | int <field-name>}0+
    {<field-type> <field-name>}0+
```

Interpretation of NTURI under the "pva" scheme

The following describes how the fields of the NTURI must be interpreted when the scheme is "pva":

scheme
The scheme name must be given. For the pva scheme, the scheme name is "pva". The pva scheme implies but does not require use of the pvAccess protocol.

authority
If given, then the IP name or address of an EPICS network pvAccess or channel access server.

path
The path gives the channel from which data is being requested.

query
A name value system for passing parameters. The types of the argument value MUST be drawn from the following restricted set of scalar types: double, int, or string.

<field-type>
Zero or more pvData Fields whose type are not defined until runtime, may be added to an NTURI by an agent creating an NTURI. This is the mechanism by which complex data may be sent to a channel. For instance a table of magnet setpoints.

The channel name given in the path MAY BE the name of an RPC channel. In that case, it's important to note that this specification makes no normative statement about where in the NTURI is encoded the name of the entity *about which* the RPC service is being called. For instance, an archive service, that gives the historical values of channels, may advertise itself as being on a single channel called say "archive service" (so the NTURI path field in that case would be set to "archiveservice", and in that case, the name of the EPICS channel about which archive data is wanted might well be encoded into one of the NTURI's query field parameters. Alternatively, the archive service might advertise a number of channels, each named perhaps after the channels whose historical data is being requested. For instance, a path may be "quad45:bdes;history", if that was the name of one of the channels offered by the archive service. An example of this second form is given below.

Use of NTURI may be explained by example. The following is an example client side of Channel RPC exchange, where a notional archive service, is asked for the data for a PV between two points in time. In this example, the archive service is advertising the channel name "quad45:bdes;history". Presumably, that service knows the archive history of a (second) channel, named probably, "quad45:bdes".

Construct the introspection interface (i.e. type definition) of the NTURI conformant structure that will be used to make requests to the archive service.

```
// Construct an NTURI for making a request to a service that understands
// query arguments named "starttime" and "endtime".
FieldCreate fieldCreate = FieldFactory.getFieldCreate();
Structure queryStructure = fieldCreate.createStructure(
    new String[] {"starttime", "endtime"},
    new Field[] { fieldCreate.createScalar(ScalarType.pvString),
                  fieldCreate.createScalar(ScalarType.pvString)});
Structure uriStructure =
    fieldCreate.createStructure("epics:nt/NTURI:1.0",
         new String[] { "path", "query" },
         new Field[] { fieldCreate.createScalar(ScalarType.pvString),
                       queryStructure } );
```

Populate our uriStructure (conformant to NTURI) with a specific request.

```
// Get a EasyPVA singleton.
EasyPVA easyPVA = EasyPVAFactory.get();

// Construct an NTURI with which to ask for the archive data of quad45:bdes
PVStructure request = PVDataFactory.getPVDataCreate().
        createPVStructure(uriStructure);
request.getStringField("path").put("quad45:bdes;history");
PVStructure query = request.getStructureField("query");
query.getStringField("starttime").put("2011-09-16T02.12.55");
query.getStringField("endtime").put("2011-09-16T10.01.03");

// Ask for the data, using the NTURI
PVStructure result = easyPVA.createChannel(request.getStringField("path").get()).
↪createRPC().request(request);
if ( result != null )
    System.out.println("The URI request structure:\n" + request
                +"\n\nResulted in:\n" + result);
```

The server side is not illustrated, but clearly its code would have registered a number of ChannelRPC services, each named after the PV whose historical data it offered.

NTNameValue

NTNameValue is the EPICS V4 Normative Type that describes a system of name and scalar values.

Use cases: In a school, a single NTNamedValue might describe the grades from a number of classes for one student.

```
NTNameValue :=

structure
    string[]     name
    scalar_t[]   value
    string       descriptor   :opt
    alarm_t      alarm        :opt
    time_t       timeStamp    :opt
```

where:

name
: The keys associated with the 'value`field. Each element of`name`identifies the same indexed element of the`value` field, using a string label.

value
: The data values, each element of which is associated with the correspondingly indexed element of the `name` field.

Each name (or "key") in the array of names, MUST be interpreted as being associated with its same indexed element of the `value` array.

NTTable

NTTable is the EPICS V4 Normative Type suitable for column-oriented tabular datasets.

An NTTable is made up of a number of arrays. Each array can be thought of as a column. Each array MUST be of a scalar type and all the arrays MUST be of the same length. Each array may be of a different scalar type. The set of the *i*th array members of all the columns make up one row, or n-tuple. The number of elements of `labels` MUST be equal to the number of fields of `value`.

Use case examples: a table of the Twiss parameters of all the lattice elements in an accelerator section. Another example, where the columns might vary call-to-call to an RPC setting, would be that of an EPICS V4 SQL database service. In that example one NTTable returned by the service would contain the tabular results of a SQL SELECT, essentially a recoded JDBC or ODBC ResultSet - see the *rdbservice*.

```
NTTable :=

structure
    string[]   labels              // Very short text describing each field below, i.e.␣
↪column labels
    structure  value
        {scalar_t[]  colname}0+ // 0 or more scalar array instances, the column values.
    string     descriptor  : opt
    alarm_t    alarm       : opt
    time_t     timeStamp   : opt
```

where:

labels
: The table column headings are given by the `labels` field. Each column heading given as one element of the array of strings.

value
: The data of the table are encoded in a structure named `value`. The columnar data field is named "value" (rather than, for instance, "columndata") so that the primary field of the type is named the same for all Normative Types. That helps general purpose clients identify the primary field of any Normative Type instance.

Interpretation

An NTTable instance represents a table of data. The column data is given in scalar arrays in the structure field `value`, and the column headings are given in field `labels`. Each / scalar array field of `value` contains the data for the column corresponding to the same indexed element of the `labels` field. Agents SHOULD use the elements of `labels` as the column headings. *There is no normative requirement that the field names of ``value`` match the strings in ``labels``.*

Note that the above description is given in terms of a table and its columns, but there is nothing specifically columnar about how this data may be rendered. A user may choose to print the fields row wise if, for instance, if there are many fields in `value`, but each has only length 1 or 2. For example, if one wanted to give all the scalar data related to one device, then one might use an NTTable rendered in such a way.

Validation

The number of *scalar_t[]* fields in the value structure, and the length of `labels` MUST be the same. All *scalar_t[]* fields in the `value` structure MUST have the same length, which is the number of "rows" in the table.

NTAttribute

NTAttribute is the EPICS V4 Normative Type for a named attribute of any type. It is is essentially a key-value pair which optionally can be tagged with additional strings.

This allows, for example, a collection of attributes to be queried on the basis of attribute name or tags.

```
NTAttribute :=

structure
    string    name
    any       value
    string[]  tags          : opt
    string    descriptor    : opt
    alarm_t   alarm         : opt
    time_t    timeStamp     : opt
```

where:

name
: The name of the attribute. The "key" of the key-value pair.

value
: The value of the attribute. The "value" of a key-value pair.

tags
: Additional tags that an attribute can carry.

1.18.10 Specific Normative Types

The "Specific Normative Types" below are types oriented towards application-level scientific and engineering use cases. Compare to *General Normative Types* defined above. The currently defined types are each described in a section below.

Unless otherwise stated:

- Times MUST be in seconds
- Frequencies MUST be in Hz.

NTMultiChannel

NTMultiChannel is an EPICS V4 Normative Type that aggregates an array of values from different EPICS Process Variable (PV) channel sources, not necessarily of the same type, into a single variable.

```
NTMultiChannel :=

structure
    anyunion_t[]  value              // The channel values
    string[]      channelName        // The channel names
```

(continues on next page)

(continued from previous page)

```
    string        descriptor         :opt
    alarm_t       alarm              :opt
    time_t        timeStamp          :opt
    int[]         severity           :opt
    int[]         status             :opt
    string[]      message            :opt
    long[]        secondsPastEpoch   :opt
    int[]         nanoseconds        :opt
    int[]         userTag            :opt
```

where:

value
The value from each channel.

channelName
The name of each channel.

alarm
The alarm associated with the NTMultiChannel itself. `severity`, `status`, and `message` show the alarm for each channel.

timeStamp
The timestamp associated with the NTMultiChannel itself. `secondsPastEpoch`, `nanoseconds` and `userTag` show the timestamp for each channel.

severity
The alarm severity associated with each channel.

status
The alarm status associated with each channel.

message
The alarm message associated with each channel.

secondsPastEpoch
The `secondsPastEpoch` field of the timestamp associated with each channel.

nanoseconds
The `nanoseconds` field of the timestamp associated with each channel.

userTag
The `userTag` field of the timestamp associated with each channel.

NTNDArray

NTNDArray is an EPICS Version 4 Normative Type designed to encode data from detectors and cameras, especially areaDetector applications. The type is heavily modeled on areaDetector's NDArray class. One NTNDArray gives one frame.

The definition of NTNDArray in full is:

```
NTNDArray :=

structure
    value_t       value
    codec_t       codec
```

(continues on next page)

(continued from previous page)

```
    long          compressedSize
    long          uncompressedSize
    dimension_t[] dimension
    int           uniqueId
    time_t        dataTimeStamp
    NTAttribute[] attribute
    string        descriptor  :opt
    alarm_t       alarm       :opt
    time_t        timeStamp   :opt
    display_t     display     :opt
```

The meaning of the above fields, the definition of *value_t* and of *dimension_t* and the additional requirements for NDAttribute are described below. To simplify this the NTNDArray can be regarded as being composed of the following parts:

```
NTNDArray :=

structure
    Image data and codec
    Data sizes
    Dimensions
    Unique ID and data timestamp
    Attributes
    Optional fields
```

Each of these will be discussed separately.

Image data and codec

The *Image data and codec* parts of an NTNDArray are composed of the following fields:

```
value_t value // Image data
codec_t codec // Codec
```

where:

value
: An array which encodes an N-dimensional array containing the data for the image itself.

codec
: Information on the how the data in value encodes the N-dimensional array.

A *value_t* is implemented as a pvData Field of type "union" with the following form:

```
value_t:=

union
    boolean[] booleanValue
    byte[]    byteValue
    short[]   shortValue
    int[]     intValue
    long[]    longValue
    ubyte[]   ubyteValue
```

(continues on next page)

(continued from previous page)

```
    ushort[]  ushortValue
    uint[]    uintValue
    ulong[]   ulongValue
    float[]   floatValue
    double[]  doubleValue
```

A *codec_t* is implemented as a pvData Field of type "structure" of type ID "codec_t" with the following form:

```
codec_t :=

structure
    string name
    any    parameters
```

where:

name
The encoding scheme, e.g. the codec in the case of compressed data.

parameters
Any additional information required to interpret the data.

The `value` field stores a scalar array of one of the scalar types permitted by the definition of `value` above whose value MUST represent an N-dimensional scalar array of one of the permitted scalar types whose dimensions are given by the `dimension` field (see below). Note that the scalar type of the array stored in `value` MAY be different from that of the array it represents.

The `codec` field is a structure which describes how the N-dimensional scalar array is represented by the value of the scalar array stored in the `value` field.

The `name` field of the `codec` field (`codec.name`) is a string which identifies the scheme by which the data in `value` is encoded, such as an algorithm used to compress the data. If it is not the empty string, the value of the `codec.name` field SHOULD be namespace qualified.

The `parameters` field of the `codec` field (`codec.parameters`) is a field which contains any additional information required to interpret the data in `value`. The format and meaning of `codec.parameters` is `codec.name`-dependent.

When the value of the `codec.name` field is the empty string the data in `value` MUST represent an N-dimensional array of the same scalar type as the scalar array stored in `value` whose dimensions are given by the `dimension` field. The elements of the array stored in `value` MUST be the elements of the N-dimensional array laid out in row major order. In this case the length of the `value` array SHOULD equal the product of the dimensions and MUST be greater than or equal to it.

When the `codec.name` field value is not the empty string the interpretation of the data in the `value` field is dependent on the `codec` field. Any requirements on the type or length of the array stored in the `value` field are `codec`-dependent.

Any endianness information associated with a compression algorithm or other encoding SHOULD be encoded via the `codec` field, either through the `codec.name` or `codec.parameters` fields.

Similarly any information required to determine the scalar type of the N-dimensional array when the value of `codec.name` field is non-empty SHOULD also be encoded in the `codec` field.

Except for the above requirements, the meaning of the `codec` field, beyond the case of the empty `codec.name` string, is not currently specified.

Data sizes

The *Data sizes]* part of an NTNDArray is composed of the following fields:

```
long compressedSize
long uncompressedSize
```

where:

compressedSize
The size of the data in bytes after any compression or other encoding.

uncompressedSize
The size of the data in bytes before any compression or other encoding.

The value of the `compressedSize` field MUST be equal to the product of the length of the scalar array field stored in the `value` field and the size of the scalar type in bytes (i.e. 1, 2, 4 or 8 for signed or unsigned byte, short, int or long respectively, 1 for boolean, 4 for float and 8 for double).

The value of the `uncompressedSize` field MUST be equal to the product of the value of the `size` field of each element in the structure array `dimension` field (described below) and the size in bytes of the scalar type of the scalar array represented by `value`. If the number of elements of the `dimension` field is 0 the value of the `uncompressedSize` MUST be 0.

Dimensions

The *Dimensions* part of an NTNDArray is composed of the `dimension` field

```
dimension_t[] dimension
```

A *dimension_t* is implemented as a pvData Field of type "structure" of type ID "dimension_t" with the following form:

```
dimension_t :=

structure
    int     size
    int     offset
    int     fullSize
    int     binning
    boolean reverse
```

where:

size
The number of elements in this dimension of the array.

offset
The offset in this dimension relative to the origin of the original data source.

fullSize
The number of elements in this dimension of the the original data source.

binning
The binning (pixel summation, 1=no binning) in this dimension relative to original data source source.

reverse
The orientation (false=normal, true=reversed) in this dimension relative to the original data source source.

The number of elements in the value of the `dimension` field MAY be 0. A client SHOULD check for this case and take appropriate action.

If an NTNDArray represents a subregion of a larger region of interest of an original image, its `offset`, `binning` and `reverse`field values SHOULD be relative to the original image and its `fullSize` field value SHOULD be the size of the original.

dimension_t is analogous to NDDimension_t in areaDetector.

Unique ID and data timestamp

The *Unique ID and data timestamp* parts of an NTNDArray are composed of the following fields:

```
int     uniqueId
time_t  dataTimeStamp
```

where:

uniqueId
: A number that SHOULD be unique for all NTNDArrays produced by a source after it has started.

dataTimeStamp
: Timestamp of the data.

The value of `dataTimeStamp` MAY be different from that of the (optional) `timeStamp` field below.

The `uniqueId` and `dataTimeStamp` fields of NTNDArray correspond to the uniqueId and timeStamp fields respectively of an NDArray.

NTNDArray attributes

The *Attributes* part of an NTNDArray is composed of the field:

```
NTAttribute[] attribute
```

where *NTAttribute* is as defined by this standard, but is extended in this case as follows:

```
NTAttribute :=

structure
    string    name
    any       value
    string[]  tags          : opt
    string    descriptor
    alarm_t   alarm         : opt
    time_t    timeStamp     : opt
    int       sourceType
    string    source
```

where:

sourceType
: The origin of the attribute

```
NDAttrSourceDriver   = 0,   /** Attribute is obtained directly from driver */
NDAttrSourceParam    = 1,   /** Attribute is obtained from an asyn parameter␣
↪library */
NDAttrSourceEPICSPV  = 2,   /** Attribute is obtained from an EPICS PV */
NDAttrSourceFunct    = 3    /** Attribute is obtained from a user-specified␣
↪function  */
```

source
: The source string of this attribute.

Note that the optional descriptor field of *NTAttribute* is mandatory for attributes of an NTNDArray.

NTAttribute here is extended by the addition of the `sourceType` and `source` fields. `source` is a string which gives the origin of the attribute according to the value of the integer `sourceType` field as follows:

- For a `sourceType` of value `NDAttrSourceDriver` the `source` string SHOULD be the empty string.
- For a `sourceType` of value `NDAttrSourceParam` the `source` string SHOULD be the name of the asyn parameter from which the attribute value was obtained.
- For a `sourceType` of value `NDAttrSourceEPICSPV` the `source` string SHOULD be the name of the EPICS PV from which the attribute value was obtained.
- For a `sourceType` of value `NDAttrSourceFunct` the `source` string SHOULD be the name of the user function from which the attribute value was obtained.

The extension of *NTAttribute* is analogous to NDAttribute in areaDetector. The `name`, `descriptor`, `sourceType` and `source` fields correspond to the pName, pDescription, sourceType, pSource members of an NDAttribute respectively.

The attributes themselves are not defined by this standard.

For areaDetector applications the `attribute` field encodes the linked list of NDAttributes in an NDArray.

[Note: areaDetector currently defines two integer attributes, colorMode and bayerPattern, with descriptions “Color mode” and “Bayer pattern” respectively:

colorMode
: An attribute that describes how an N-d array is to be interpreted as an image, taking one of the values in this enumeration:

```
NDColorModeMono    = 0,    /** Monochromatic image */
NDColorModeBayer   = 1,    /** Bayer pattern image,
                               1 value per pixel but with color filter on detector */
NDColorModeRGB1    = 2,    /** RGB image with pixel color interleave,
                               data array is [3, NX, NY] */
NDColorModeRGB2    = 3,    /** RGB image with row color interleave,
                               data array is [NX, 3, NY]  */
NDColorModeRGB3    = 4,    /** RGB image with plane color interleave,
                               data array is [NX, NY, 3]  */
NDColorModeYUV444  = 5,    /** YUV image, 3 bytes encodes 1 RGB pixel */
NDColorModeYUV422  = 6,    /** YUV image, 4 bytes encodes 2 RGB pixel */
NDColorModeYUV411  = 7     /** YUV image, 6 bytes encodes 4 RGB pixels */
```

bayerPattern
: An attribute valid when colorMode is NDColorModeBayer providing additional information required for the interpretation of an N-d array as an image in this case, taking one of the values in this enumeration:

```
NDBayerRGGB       = 0,    /** First line RGRG, second line GBGB... */
NDBayerGBRG       = 1,    /** First line GBGB, second line RGRG... */
NDBayerGRBG       = 2,    /** First line GRGR, second line BGBG... */
NDBayerBGGR       = 3     /** First line BGBG, second line GRGR... */
```

Other areaDetector attributes are user-defined.]

NTContinuum

NTContinuum is the EPICS V4 Normative Type used to express a sequence of point values in time or frequency domain. Each point has N values (N>=1) and an additional value which describes the index of the list. The additional value is carried in the `base` field. The `value` field carries the values which make up the point in index order.

An additional `units` field gives a units string for the N values and the additional value.

```
NTContinuum :=

structure
    double[]   base
    double[]   value
    string[]   units
    string     descriptor    :opt
    alarm_t    alarm         :opt
    time_t     timeStamp     :opt
```

The number of values in a point must be derived as:

Nvals = len(value)/len(base)

And the following invariant must be preserved:

len(units)-1 == Nvals

For points (A_i, B_i, C_i) for indices i = 1, 2, 3 the `value` array is:

[A_1, B_1, C_1, A_2, B_2, C_2, A_3, B_3, C_3]

NTHistogram

NTHistogram is the EPICS V4 Normative Type used to encode the data and representation of a (1 dimensional) histogram. Specifically, it encapsulates frequency binned data.

For 2d histograms (i.e. both x and y observations are binned) and n-tuple data (e.g. land masses of different listed countries) see NTMatrix or NTTable.

```
NTHistogram :=

structure
    double[]   ranges                    // The start and end points of each bin
    (short[] | int[] | long[])  value    // The frequency count, or otherwise value, of␣
↪each bin
    string     descriptor    :opt
    alarm_t    alarm         :opt
    time_t     timeStamp     :opt
```

Interpretation

One NTHistogram gives the information required to convey a histogram representation of some underlying observations. It does not convey the values of each of the observations themselves.

The number of bins is given by the length of the `value` array. `ranges` indicates the low value and high value of each bin. The range for *bin(i)* is given by *ranges(i)* to *ranges(i+1)*. Specifically, since we want end points of both the first bin and last bin included, all bin intervals except the last one, MUST be *right half open*; from that bin's low value *ranges(i)* (included) to that bin's high value *ranges(i+1)* (excluded). The last bin MUST be fully *open* (low and high value included).

A log plot histogram (in which the independent variable x is binned on a log scale), would be communicated using a range array of decades (1.0E01, 1.0E02, 1.0E03 etc).

Validation

The array length of `ranges` MUST be the array length of `value` + 1.

NTAggregate

NTAggregate is the EPICS V4 Normative Type to compactly convey data which combines several measurements or observation. NTAggregate gives simple summary statistic *bib:agg* about the central tendency and dispersion of a set of data points.

Use cases: for instance, an NTAggregate could be used to summarize the value of one beam position offset reading over some number of pulses (N). It also includes the time range of the sampled points, so it could be used for time domain rebasing. For instance, an FPGA sending data at 10KHz, and you want to display its output, but you don't want to display at the native rate. Also, it could be used for transmitting or storing compressed archive data.

NTAggregate doesn't cover the shape of a distribution so it only reasonably helps you do symmetrical distributions (no skewness or kurtosis), and it doesn't include any help for indicating the extent of dependency on another variable (correlation).

```
NTAggregate :=

structure
    double     value                // The center point of the observations,
                                    // nominally the mean.
    long       N                    // Number of observations
    double     dispersion     :opt  // Dispersion of observations;
                                     // nominally the Standard Deviation or RMS
    double     first          :opt  // Initial observation value
    time_t     firstTimeStamp :opt  // Time of initial observation
    double     last           :opt  // Final observation value
    time_t     lastTimeStamp  :opt  // Time of final observation
    double     max            :opt  // Highest value in the N observations
    double     min            :opt  // Lowest value in the N observations
    string     descriptor     :opt
    alarm_t    alarm          :opt
    time_t     timeStamp      :opt
```

where:

value
The summary statistic of the set of observations conveyed by this NTAggregate. For instance their arithmetic mean.

N
The number of observations summarized by this NTAggregate.

dispersion
The extent to which the observations are centered around the `value`. For instance, if the `value` contains a mean, then the dispersion may be the variance or the standard deviation. The `descriptor` should indicate which.

first
The value of the temporally first observation conveyed by this NTAggregate.

firstTimeStamp
The time of observation of the temporally first observation conveyed by this NTAggregate.

last
The value of the temporally final observation conveyed by this NTAggregate.

lastTimeStamp
The time of observation of the temporally final observation conveyed by this NTAggregate.

max
The numerically largest value in the set of observations conveyed by this NTAggregate.

min
The numerically smallest value in the set of observations conveyed by this NTAggregate.

Interpretation

One NTAggregate instance describes some number (given by N) of observations. If firstTimeStamp and lastTimeStamp are given, then the N observations MUST have been taken over the period of time specified. If first, last, max or min are given, they MUST refer to the actual values of the N observations being summarized.

The `value` field value computed by server agents may be the arithmetic mean of the observation data being summarized by this NTAggregate, but NTAggregate does not normatively define that. Other measures of mean (geometric, harmonic) may be assigned. Indeed other measures of central tendency may be used. The interpretation to give an instance of an NTAggregate SHOULD be conveyed in the `descriptor`.

Where dispersion is a measure of the standard deviation, which estimator of the standard deviation [1/N or 1/(N-1)] was used, is also not defined normatively.

1.18.11 Appendix A: Possible Future Additions to this Specification

NTUnion

NTUnion would be a Normative Type for interoperation of essentially any data structure, plus description, alarm and timestamp fields.

```
NTUnion :=

structure
    anyunion_t   value
    string       descriptor       :opt
    alarm_t      alarm            :opt
    time_t       timeStamp        :opt
```

NTScalarMultiChannel

NTScalarMultiChannel is an EPICS V4 Normative Type that aggregates an array of values from different EPICS Process Variable (PV) channel sources of the same scalar type into a single variable.

Use cases: In a particle accelerator, a single NTScalarMultiChannel might include the data of a number of Beam Position Monitors' X offset values, or of a number of quadrupoles' desired field values.

```
NTScalarMultiChannel :=

structure
    scalar_t[]    value              // The channel values
    string[]      channelName        // The channel names
    string        descriptor         :opt
    alarm_t       alarm              :opt
    time_t        timeStamp          :opt
    int[]         severity           :opt
    int[]         status             :opt
    string[]      message            :opt
    long[]        secondsPastEpoch   :opt
    int[]         nanoseconds        :opt
    int[]         userTag            :opt
```

where:

value
The value from each channel.

channelName
The name of each channel.

alarm
The alarm associated with the NTScalarMultiChannel itself. `severity`, `status`, and `message` show the alarm for each channel.

timeStamp
The timestamp associated with the NTScalarMultiChannel itself. `secondsPastEpoch`, `nanoseconds` and `userTag` show the timestamp for each channel.

severity
The alarm severity associated with each channel.

status
The alarm status associated with each channel.

message
The alarm message associated with each channel.

secondsPastEpoch
The `secondsPastEpoch` field of the timestamp associated with each channel.

nanoseconds
The `nanoseconds` field of the timestamp associated with each channel.

userTag
The `userTag` field of the timestamp associated with each channel.

1.18.12 Appendix B: Normative Type Identifiers

This Appendix describes the Normative Type Identifiers of the abstract data types defined by this document. These are the strings which identify the type carried by a structure. In the pvAccess binding (which is at present the only one implemented for EPICS V4), the type ID of the structure MUST carry one of these identifier strings. In doing so, the structure instance declares itself to conform to the corresponding definition carried in this specification document.

The syntax of the Normative Type identifier is:

```
namespacename/typename:versionnumber
```

The Normative Type Identifier "Namespace Name" part, is:

```
epics:nt
```

The Normative Type Identifier "Type Name" and version number parts corresponding to *this draft* of the Normative Types Document (this document), MUST be valued as following:

Table 2: Type Names that may be used in the Type Name part of a Normative Type Identifier of an EPICS V4 Normative Type in the namespace of this draft of the Normative Types specification

Type Name	Version	Depends on	Short Description
NTScalar	1.0	(none)	A single scalar value.
NTScalarArray	1.0	(none)	An array of scalar values of some single type.
NTEnum	1.0	(none)	An enumeration list and a value of that enumeration.
NTMatrix	1.0	(none)	A real number matrix.
NTURI	1.0	(none)	A structure for encapsulating a Uniform Resource Identifier (URI).
NTNameValue	1.0	(none)	An array of scalar values where each element is named.
NTTable	1.0	(none)	A table of scalars, where each column may be of different scalar array type
NTAttribute	1.0	(none)	A key-value pair, with optional string tags, where the value is of any type.
NTMultiChannel	1.0	(none)	An array of PV names, their values, and metadata.
NTNDArray	1.0	NTAttribute 1.0	A pixel and metadata type, designed to encode a frame of data from detectors and cameras.
NTContinuum	1.0	(none)	Expresses a sequence of data points in time or frequency domain.
NTHistogram	1.0	(none)	An array of real number intervals, and their frequency counts. Expresses a 1D histogram.
NTAggregate	1.0	(none)	A mean value, standard deviation, and other metadata. Expresses the central tendency and dispersion of a set of data points.

For example, the type ID of a structure describing an NTScalar, must be valued "epics:nt/NTScalar:1.0". The type ID of a structure describing an NTNDArray, must be valued "epics:nt/NTNDArray:1.0".

Following drafts of this document MAY well correspond to the same Namespace Name and Type Names as used in this draft. Also note that the same namespace may well be used for a different collection of types or Type Names, as this document matures.

1.18.13 Bibliography

[bib:pvdata]
EPICS V4 Documentation page, Programmers' Reference Documentation section (pvData).

[bib:pvaccess]
V4 Documentation page, Programmers' Reference Documentation section (pvAccess).

[bib:epicsrecref]
EPICS Reference Manual, Philip Stanley, Janet Anderson, Marty Kraimer, APS, https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14.

[bib:epicsappdev]
EPICS Input / Output Controller (IOC) Application Developer's Guide Marty Kraimer, APS, 1994, http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/.

bib:agg
Aggregate data, Wikipedia article, http://en.wikipedia.org/wiki/Aggregate_data.

bib:rdbservice
rdbService, example EPICS V4 service, https://github.com/epics-base/exampleJava/tree/master/src/services/rdbService.

bib:uri
Uniform Resource Identifiers (URI): Generic Syntax, http://www.ietf.org/rfc/rfc2396.txt.

1.19 EPICS 7, pvAccess and pvData

Tags: developer advanced

pvAccess, pvData and other related modules have been introduced into EPICS to add support for structured data. Let us look into the reasons, and also at some use cases for the capabilities to handle structured data.

EPICS has its roots in process control. In typical process control applications, process variables are scalar data items. Transporting the process variables efficiently has priority over handling sophisticated constructs. Only a limited set of data is sufficient to describe the process data: timestamp, alarm status, display information and engineering units. This kind of simple interfaces make it possible to build general-purpose tools for manipulating the data, and also enables the low-level units to interoperate without big overhead or having to customize the applications whenever a new structure is introduced.

However, in more complex applications like data acquisition in scientific experiments, having only scalar values and limited metadata becomes a limiting factor. For instance, when (camera) images are transported over the network, more complex metadata is required to interpret and display the image; what are the dimensions of the picture in pixels, how many data bits are required to present a single pixel, what is the encoding, and many other parameters.

Even further, when it is required to represent more abstract entities, single values or primitive waveforms are not suitable for these tasks.

It is possible work around these limitations to some extent. One can define several process variables and combine these in a higher-level application. This has been done in many packages, for instance in accelerator physics applications like beam steering, by building an abstraction layer on top the simple process variables.

While workarounds are possible, they have many drawbacks. To begin with, it is very difficult to ensure interoperability of applications that have been built in this way. The logic gets dispersed in various layers of the software stack and applications cannot take advantage of what has been implemented in other parts of the system. For instance, an archiver cannot store data entities that have been defined in a physics application, nor can a general-purpose GUI client display them.

Also, sharing of data is difficult, even between colleagues in the same organization or project, not even to talk about making the data useful outside of the organization.

This situation leads to limited functionality and also every project has to build a set of site-specific applications from scratch.

1.20 Overview of pvData implementation

pvData (Process Variable Data) is a part of the EPICS (7 and above) core software. It is a run-time type system with serialization and introspection for handling of structured data. It implements the data management system to which the other related components like pvAccess, pvDatabase, etc. interface.

pvData is conceptually somewhat similar to Google Protocol Buffers (PB, see [1]), however an important difference is that pvData type and structure information is exchanged run-time, while PB depends on precompiled stubs on each side.

pvData defines and implements an efficient way to store, access, and communicate memory resident data structures. The following attributes describe the design goals of pvData:

- efficiency
  - Small memory footprint, low CPU overhead, and concise code base.
- simple but powerful structure concept
  - pvData has four types of data **fields**: scalar, scalarArray, structure, and structureArray. A **scalar** can be one of the following scalar types: Boolean, Byte, Short, Int, Long, U(nsigned) Byte, Unsigned Short, Unsigned Int, Unsigned Long, Float, Double, and String. A **scalarArray** is a one-dimensional array with the element type being any of the scalar types. A **structure** is an ordered set of fields where each field has a name and type. A **structureArray** is an array of similar structures. Since a field can be a structure, complex structures can be created.
- structure and data storage separation
  - pvData defines separate introspection and data interfaces. The introspection interfaces provide access to immutable objects, which allows introspection instances to be freely shared. The introspection interface for a process variable can be accessed without requiring access to the data.
- data transfer optimization
  - The separation of introspection and data interfaces allows for efficient network data transfer. When a client connects to a PV, introspection information is passed from server to client so that each side can create a data instance. The payload data is transferred between these instances. The data that is transferred over the network does not have to be self-describing since each side has the introspection information.
- data access standardization
  - Client code can access pvData via the introspection and data interfaces. For "well known" data, e.g. image data, specialized interfaces can be provided without requiring any changes to the core software. There exists a separate definition of standard data types, called Normative Types. For example, a normative type for image data is called NTNDArray.
- memory resident
  - pvData only defines memory resident data.

pvData is intended for use by pvAccess client software, as an interface between client and network, or network and server, as well as an interface between server and IOC database. Since it is a system-agnostic interface to data, it could also be used by other systems and is easy to convert between different storage formats. A high-level physics application can manipulate data as pvData structures, the data can made available to network clients by a pvAccess server like

qsrv that is included in an EPICS IOC to serve process variables, or some special-purpose server, serving for example calibration data from a suitable data storage like a database.

1.21 PVData structure definition

This section describes pvData structures in a metalanguage. The metalanguage is used for documentation; there are no parsers or a strict formal description. The metalanguage is used to describe both introspection interfaces and data interfaces.

1.21.1 Definitions

PVData supports structured data. All data appears via top-level structures. A structure has an ordered set of fields where each field is defined as follows:

type fieldName *value* // comment

where *value* is present for data objects and // indicates that the the rest of the line is a comment.

type is one of **scalar**, **scalarArray**, **structure**, or **structureArray**. These types are defined in more details in the following paragraphs.

1.21.2 scalar

A scalar field can be of any of the following primitive types:

boolean

Has the value “true” or “false”.

byte

An 8 bit signed integer.

short

An 16 bit signed integer.

int

An 32 bit signed integer.

long

An 64 bit signed integer.

ubyte

An 8 bit unsigned integer.

ushort

An 16 bit unsigned integer.

uint

An 32 bit unsigned integer.

ulong

An 64 bit unsigned integer.

float

A IEEE float.

double

A IEEE double.

string

An immutable string.

1.21.3 scalarArray

A scalarArray field is an array of any of the scalar types.

boolean[]

byte[]

short[]

int[]

long[]

ubyte[]

ushort[]

uint[]

ulong[]

float[]

double[]

string[]

1.21.4 structure

A structure field has the definition:

structure *fieldName*

fieldDef

...

or

xxx_t *fieldName*

// if data object then following appears

fieldDef

...

For structure fieldName each *fieldDef* must have a unique fieldName within the structure.

For "xxx_t fieldName", xxx_t must be a previously defined structure of the form:

structure *xxx_t* ...

1.21.5 structureArray

A structureArray field has the definition:

structure[] *fieldName* structureDef ...

or

xxx_t[] fieldName

Thus a structure array is an array where each element is a structure but all elements of the array have the same structure and also the same introspection interface. For introspection the structureDef appears once without any data values.

The above is used to describe introspection objects. Data objects are described in a similar way but each scalar field and each array field has data values. The definition of the data values depends on the type. For scalars the data value is whatever is valid for the type.

boolean

The value must be true or false

byte,...ulong

Any valid integer or hex value, e.g. 3 and 0x0ff are valid values

float,double

Any valid integer or real e.g. 3, 3.0, and 3e0 are valid values

string

The value can be an alphanumeric value or any set of characters enclosed in "" Within quotes a quote is expressed as \" Examples are aValue "a value" "a" xxx" are valid values.

For scalar arrays the syntax is:

= [value,...,value]

where each value is a valid scalar data value depending on the type. Thus it is a comma separated set of values enclosed in square brackets: [] White space is permitted surrounding each comma.

Examples

Having defined the following base structure:

```
structure  timeStamp_t
  long secondsPastEpoch
  int nanoSeconds
  int userTag
```

it can be used to define further structures:

```
structure  scalarDoubleExample // introspection object
  double value
  timeStamp_t timeStamp
```

which would correspond to:

```
structure scalarDoubleExample
  double value
  structure timeStamp
    long secondsPastEpoch
```

(continues on next page)

(continued from previous page)

```
    int nanoSeconds
    int userTag
```

The following corresponding **data** object can then be defined:

```
structure scalarDoubleExample // data object
  double value 1.0
  timeStamp_t timeStamp
    long secondsPastEpoch 1531389047
    int nanoSeconds 247000000
```

Also, if the following interface is defined:

```
structure point_t
  double x
  double y
```

the following uses become possible (among others):

```
structure lineExample
  point_t begin
  point_t end

structure pointArrayExample
  point_t[] points
```

filling in the details, they look like:

```
structure lineExample
  structure begin
    double x
    double y
  structure end
    double x
    double y
```

and

```
structure pointArrayExample
  structure[] points
    structure point
      double x
      double y
```

And the corresponding **data** objects could look like this:

```
structure lineExample
  point_t begin
    double x 0.0
    double y 0.0
  point_t end
    double x 10.0
    double y 10.0
```

(continues on next page)

(continued from previous page)

```
structure pointArrayExample
  point_t[] value
    structure point
      double x 0.0
      double y 0.0
    structure point
      double x 10.0
      double y 10.0
```

References:

1. Google Protocol Buffers: http://code.google.com/apis/protocolbuffers/
2. Normative Types Specification

1.22 IOC Access Security

Tags: developer advanced

Table of Contents

1.22.1 Features

Access security protects IOC databases from unauthorized Channel Access or pvAccess Clients. Access security is based on the following:

Who
User id of the client(Channel Access/pvAccess).

Where
Host id where the user is logged on. This is the host on which the client exists. Thus no attempt is made to see if a user is local or is remotely logged on to the host.

What
Individual fields of records are protected. Each record has a field containing the Access Security Group (ASG) to which the record belongs. Each field has an access security level, either ASL0 or ASL1. The security level is defined in the record definition file (.dbd). Thus the access security level for a field is the same for all record instances of a record type.

When
Access rules can contain input links and calculations similar to the calculation record.

Limitations

An IOC database can be accessed only via pvAccess, Channel Access or the ioc (or vxWorks) shell. It is assumed that access to the local IOC console is protected via physical security, and that network access is protected via normal networking and physical security methods.

No attempt has been made to protect against the sophisticated saboteur. Network and physical security methods must be used to limit access to the subnet on which the IOCs reside.

Definitions

This document uses the following terms:

ASL
Access Security Level.

ASG
Access Security Group

UAG
User Access Group

HAG
Host Access Group

1.22.2 Quick Start

In order to "turn on" access security for a particular IOC the following must be done:

- Create the access security file.
- IOC databases may have to be modified
  - Record instances may have to have values assigned to field ASG. If ASG is null the record is in group DEFAULT.

  - Access security files can be reloaded after iocInit via a subroutine record with asSubInit and asSubProcess as the associated subroutines. Writing the value 1 to this record will cause a reload.
  - The startup script must contain the following command before iocInit.

```
asSetFilename("/full/path/to/accessSecurityFile")
```

- The following is an optional command.

```
asSetSubstitutions("var1=sub1,var2=sub2,...")
```

The following rules decide if access security is turned on for an IOC:

- If asSetFilename is not executed before iocInit, access security will never be started.
- If asSetFile is given and any error occurs while first initializing access security, then all access to that ioc is denied.
- If after successfully starting access security, an attempt is made to restart and an error occurs then the previous access security configuration is maintained.

After an IOC has been booted with access security enabled, the access security rules can be changed by issuing the asSetFilename, asSetSubstitutions, and asInit. The functions asInitialize, asInitFile, and asInitFP, which are described below, can also be used.

Access Security Configuration File

This section describes the format of a file containing definitions of the user access groups, host access groups, and access security groups. An IOC creates an access configuration database by reading an access configuration file (the extension .acf is recommended). Lets first give a simple example and then a complete description of the syntax.

Simple Example

```
UAG(uag) {user1,user2}
HAG(hag) {host1,host2}
ASG(DEFAULT) {
        RULE(1,READ)
        RULE(1,WRITE) {
                UAG(uag)
                HAG(hag)
       }
}
```

These rules provide read access to anyone located anywhere and write access to user1 and user2 if they are located at host1 or host2.

Syntax Definition

In the following description:

[ ] surrounds optional elements

| separates alternatives

... means that an arbitrary number of definitions may be given.

introduces a comment line

The elements <name>, <user>, <host>, <pvname> and <calculation> can be given as quoted or unquoted strings. The rules for unquoted strings are the same as for database definitions.

```
UAG(<name>) [{ <user> [, <user> ...] }]
...
HAG(<name>) [{ <host> [, <host> ...] }]
...
ASG(<name>) [{
    [INP<index>(<pvname>)
    ...]
    RULE(<level>,NONE | READ | WRITE [, NOTRAPWRITE | TRAPWRITE]) {
        [UAG(<name> [,<name> ...])]
        [HAG(<name> [,<name> ...])]
        CALC(<calculation>)
    }
    ...
}]
...
```

Discussion

- UAG: User Access Group. This is a list of user names. The list may be empty. A user name may appear in more than one UAG. To match, a user name must be identical to the user name read by the CA client library running on the client machine. For vxWorks clients, the user name is usually taken from the user field of the boot parameters.
- HAG: Host Access Group. This is a list of host names. It may be empty. The same host name can appear in multiple HAGs. To match, a host name must match the host name read by the CA client library running on the client machine; both names are converted to lower case before comparison however. For vxWorks clients, the host name is usually taken from the target name of the boot parameters.
- ASG: An access security group. The group DEFAULT is a special case. If a member specifies a null group or a group which has no ASG definition then the member is assigned to the group DEFAULT.
- INP<index>Index must have one of the values A to L. These are just like the INP fields of a calculation record. It is necessary to define INP fields if a CALC field is defined in any RULE for the ASG.
- RULE This defines access permissions. <level> must be 0 or 1. Permission for a level 1 field implies permission for level 0 fields. The permissions are NONE, READ, and WRITE. WRITE permission implies READ permission. The standard EPICS record types have all fields set to level 1 except for VAL, CMD (command), and RES (reset). An optional argument specifies if writes should be trapped. See the section below on trapping Channel Access writes for how this is used. If not given the default is NOTRAPWRITE.
  - UAG specifies a list of user access groups that can have the access privilege. If UAG is not defined then all users are allowed.

- HAG specifies a list of host access groups that have the access privilege. If HAG is not defined then all hosts are allowed.
- CALC is just like the CALC field of a calculation record except that the result must evaluate to TRUE or FALSE. The rule only applies if the calculation result is TRUE, where the actual test for TRUE is (0.99 < result < 1.01). Anything else is regarded as FALSE and will cause the rule to be ignored. Assignment statements are not permitted in CALC expressions here.

Each IOC record contains a field ASG, which specifies the name of the ASG to which the record belongs. If this field is null or specifies a group which is not defined in the access security file then the record is placed in group DEFAULT.

The access privilege for a channel access client is determined as follows:

1. The ASG associated with the record is searched.
2. Each RULE is checked for the following:
    1. The field's level must be less than or equal to the level for this RULE.
    2. If UAG is defined, the user must belong to one of the specified UAGs. If UAG is not defined all users are accepted.
    3. If HAG is defined, the user's host must belong to one one of the HAGs. If HAG is not defined all hosts are accepted.
    4. If CALC is specified, the calculation must yield the value 1, i.e. TRUE. If any of the INP fields associated with this calculation are in INVALID alarm severity the calculation is considered false. The actual test for TRUE is .99 <result <1.01.
3. The maximum access allowed by step 2 is the access chosen.

Multiple RULEs can be defined for a given ASG, even RULEs with identical levels and access permissions. The TRAPWRITE setting used for a client is determined by the first WRITE rule that passes the rule checks.

ascheck - Check Syntax of Access Configuration File

After creating or modifying an access configuration file it can be checked for syntax errors by issuing the command:

```
ascheck -S "xxx=yyy,..." < "filename"
```

This is a Unix command. It displays errors on stdout. If no errors are detected it prints nothing. Only syntax errors not logic errors are detected. Thus it is still possible to get your self in trouble. The flag -S means a set of macro substitutions may appear. This is just like the macro substitutions for dbLoadDatabase.

IOC Access Security Initialization

In order to have access security turned on during IOC initialization the following command must appear in the startup file before iocInit is called:

```
asSetFilename("/full/path/to/access/security/file.acf")
```

If this command is not used then access security will not be started by iocInit. If an error occurs when iocInit calls asInit than all access to the ioc is disabled, i.e. no channel access client will be able to access the ioc. Note that this command does not read the file itself, it just saves the argument string for use later on, nor does it save the current working directory, which is why the use of an absolute path-name for the file is recommended (a path name could be specified relative to the current directory at the time when iocInit is run, but this is not recommended if the IOC also loads the subroutine record support as a later reload of the file might happen after the current directory had been changed).

Access security also supports macro substitution just like dbLoadDatabase. The following command specifies the desired substitutions:

```
asSetSubstitutions("var1=sub1,var2=sub2,...")
```

This command must be issued before iocInit.

After an IOC is initialized the access security database can be changed. The preferred way is via the subroutine record described in the next section. It can also be changed by issuing the following command to the vxWorks shell:

```
asInit
```

It is also possible to reissue asSetFilename and/or asSetSubstitutions before asInit. If any error occurs during asInit the old access security configuration is maintained. It is NOT permissible to call asInit before iocInit is called.

Restarting access security after ioc initialization is an expensive operation and should not be used as a regular procedure.

1.22.3 Database Configuration

Access Security Group

Each database record has a field ASG which holds a character string. Any database configuration tool can be used to give a value to this field. If the ASG of a record is not defined or is not equal to a ASG in the configuration file then the record is placed in DEFAULT.

Subroutine Record Support

Two subroutines, which can be attached to a subroutine record, are available (provided with iocCore):

```
asSubInit
asSubProcess
```

NOTE: These subroutines are automatically registered thus do NOT put a registrar definition in your database definition file.

If a record is created that attaches to these routines, it can be used to force the IOC to load a new access configuration database. To change the access configuration:

1. Modify the file specified by the last call to asSetFilename so that it contains the new configuration desired.
2. Write a 1 to the subroutine record VAL field. Note that this can be done via channel access.

The following action is taken:

1. When the value is found to be 1, asInit is called and the value set back to 0.
2. The record is treated as an asynchronous record. Completion occurs when the new access configuration has been initialized or a time-out occurs. If initialization fails the record is placed into alarm with a severity determined by BRSV.

Record Type Description

Each field of each record type has an associated access security level of ASL0 or ASL1 (default value). Fields which operators normally change are assigned ASL0, other fields are assigned ASL1. For example, the VAL field of an analog output record is assigned ASL0 and all other fields ASL1. This is because only the VAL field should be modified during normal operations.

Example:

Lets design a set of rules for a Linac. Assume the following:

1. Anyone can have read access to all fields at anytime.
2. Linac engineers, located in the injection control or control room, can have write access to most level 0 fields only if the Linac is not in operational mode.
3. Operators, located in the injection control or control room, can have write access to most level 0 fields anytime.
4. The operations supervisor, linac supervisor, and the application developers can have write access to all fields but must have some way of not changing something inadvertently.
5. Most records use the above rules but a few (high voltage power supplies, etc.) are placed under tighter control. These will follow rules 1 and 4 but not 2 or 3.
6. IOC channel access clients always have level 1 write privilege.

Most Linac IOC records will not have the ASG field defined and will thus be placed in ASG DEFAULT. The following records will have an ASG defined:

- LI:OPSTATE and any other records that need tighter control have ASG=”critical”. One such record could be a subroutine record used to cause a new access configuration file to be loaded. LI:OPSTATE has the value (0,1) if the Linac is (not operational, operational).
- LI:lev1permit has ASG=”permit”. In order for the opSup, linacSup, or an appDev to have write privilege to everything this record must be set to the value 1.

The following access configuration satisfies the above rules.

```
UAG(op) {op1,op2,superguy}
UAG(opSup) {superguy}
UAG(linac) {waw,nassiri,grelick,berg,fuja,gsm}
UAG(linacSup) {gsm}
UAG(appDev) {nda,kko}
HAG(icr) {silver,phebos,gaea}
HAG(cr) {mars,hera,gold}
HAG(ioc) {ioclic1,ioclic2,ioclid1,ioclid2,ioclid3,ioclid4,ioclid5}
ASG(DEFAULT) {
    INPA(LI:OPSTATE)
    INPB(LI:lev1permit)
    RULE(0,WRITE) {
        UAG(op)
        HAG(icr,cr)
        CALC("A=1")
    }
    RULE(0,WRITE) {
        UAG(op,linac,appdev)
        HAG(icr,cr)
```

(continues on next page)

(continued from previous page)

```
        CALC("A=0")
    }
    RULE(1,WRITE) {
        UAG(opSup,linacSup,appdev)
        CALC("B=1")
    }
    RULE(1,READ)
    RULE(1,WRITE) {
        HAG(ioc)
    }
}
ASG(permit) {
    RULE(0,WRITE) {
        UAG(opSup,linacSup,appDev)
    }
    RULE(1,READ)
    RULE(1,WRITE) {
        HAG(ioc)
    }
}
ASG(critical) {
    INPB(LI:lev1permit)
    RULE(1,WRITE) {
        UAG(opSup,linacSup,appdev)
        CALC("B=1")
    }
    RULE(1,READ)
    RULE(1,WRITE) {
        HAG(ioc)
    }
}
```

Summary of Functional Requirements

A brief summary of the Functional Requirements is:

1. Each field of each record type is assigned an access security level.
2. Each record instance is assigned to a unique access security group.
3. Each user is assigned to one or more user access groups.
4. Each node is assigned to a host access group.
5. For each access security group a set of access rules can be defined. Each rule specifies:
    1. Access security level
    2. READ or READ/WRITE access.
    3. An optional list of User Access Groups or * meaning anyone.
    4. An optional list of Host Access Groups or * meaning anywhere.
    5. Conditions based on values of process variables

Additional Requirements

Performance

Although the functional requirements do not mention it, a fundamental goal is performance. The design provides almost no overhead during normal database access and moderate overhead for the following: channel access client/server connection, ioc initialization, a change in value of a process variable referenced by an access calculation, and dynamically changing a records access control group. Dynamically changing the user access groups, host access groups, or the rules, however, can be a time consuming operation. This is done, however, by a low priority IOC task and thus does not impact normal ioc operation.

Generic Implementation

Access security should be implemented as a stand alone system, i.e. it should not be embedded tightly in database or channel access.

No Access Security within an IOC

No access security is invoked within an IOC . This means that database links and local channel access clients calls are not subject to access control. Also test routines such as dbgf should not be subject to access control.

Defaults

It must be possible to easily define default access rules.

Access Security is Optional

When an IOC is initialized, access security is optional.

pvAccess (QSRV) Specific Features

QSRV will enforce the access control policy loaded by the usual means (cf. asSetFilename()). This policy is applied to both Single and Group PVs. With Group PVs, restrictions are not defined for the group, but rather for the individual member records. The same policy will be applied regardless of how a record is accessed (individually, or through a group).

Policy application differs from CA (RSRV) in several ways:

Client hostname is always the numeric IP address. HAG() entries must either contain numeric IP addresses, or **asCheckClientIP=1** flag must be set to translate hostnames into IPs on ACF file load (effects CA server as well). This prevents clients from trivially forging "hostname". In additional to client usernames, UAG definitions may contained items beginning with "role/" which are matched against the list of groups of which the client username is a member. Username to group lookup is done internally to QSRV, and depends on IOC host authentication configuration. Note that this is still based on the client provided username string.

```
UAG(special) {
   someone, "role/op"
}
```

The "special" UAG will match CA or PVA clients with the username "someone". It will also match a PVA client if the client provided username is a member of the "op" group (supported on POSIX targets and Windows).

1.23 How to Configure Channel Access

Tags: beginner user

1.23.1 Basic Operation, One IOC on same subnet

Assume an IOC has a record `fred`, and you want to use `caget fred` or a similar CA client to read it.

When starting out with one IOC on the network, things are simple:

CA clients will by default broadcast name search requests to UDP port 5064 on the subnet. As long as the IOC is running on on any computer on that subnet, it should receive those search requests. Server and client will then establish a TCP connection, and data is exchanged.

1.23.2 Multiple IOCs on different computers, but same subnet

If running multiple IOCs, each on their own computer, on the same subnet, the basic broadcast name search will still succeed, no change necessary.

1.23.3 IOCs on different subnets

The default broadcast name search is limited to the subnet of the computer running the CA client. To reach IOCs on one or more additional subnets, the environment variable `EPICS_CA_ADDR_LIST` needs to be configured. It can list either the specific IP addresses of each IOC, or the broadcast address of their subnet. Note, however, that routers will often not forward broadcast requests, which suggests using specific IP addresses.

1.23.4 Multiple IOCs on the same computer

When starting the first IOC on a computer, it will listen to name searches on UDP port 5064. When starting a second IOC on the same computer, it will also listen to name searches on UDP port 5064. Due to limitations in most network kernels, however, only the IOC started *last* will actually receive UDP search requests that are sent to that computer, port 5064. As a workaround, you need to configure the `EPICS_CA_ADDR_LIST` to use the broadcast address of the respective subnet.

Alternatively, you can automatically set up iptables rules that will circumvent the problem. (See *How to Make Channel Access Reach Multiple Soft IOCs on a Linux Host*.)

1.23.5 Multiple IOCs on the same computer but on a different subnet

Combining the last two points results in a problem: To reach multiple IOCs on the same computer, `EPICS_CA_ADDR_LIST` must be set to the broadcast address of that computer's subnet. If the IOCs' subnet is different from the CA client's subnet however, the broadcast search packets will not usually be forwarded by the intermediate network routers.

There are several options to solve this:

Channel Access Gateway

The PV gateway, running on the subnet that has the desired IOCs, will use the broadcast address of that subnet in its `EPICS_CA_ADDR_LIST`, so it can reach all IOCs, including multiple IOCs running on the same computer, throughout that subnet. A CA client on a different subnet uses only `EPICS_CA_ADDR_LIST=ip-of-the-gateway` to directly reach the gateway, which is possible via routers.

In addition to establishing the basic connectivity, the gateway also offers IOC load reduction and it can add access security, for example limit write access.

CA Nameserver

You can run a CA Name Server in the GUI subnet which knows about the IOCs and responds to search requests; in this case you would *not* set the `EPICS_CA_ADDR_LIST` variables. This is almost equivalent to running a CA Gateway, but is slightly more robust because if the Nameserver process dies it wouldn't kill any existing connections.

UDP Broadcast Packet Relay

If you have access to a machine with a network interface on both subnets you can run a program on it called UDP Broadcast Packet Relay to forward UDP broadcast packets between the subnets. For best performance you should run it twice, once for port 5064 and again for 5065. The first one will forward CA search requests between the subnets, while the second redistributes CA beacons which help channels reconnect faster after an IOC has been turned off for some time.

1.23.6 Firewalls

Firewalls may need to be configured to pass UDP and TCP traffic on both ports 5064 and 5065.

The Channel Access Reference Manual provides a lot more detail.

1.24 How to find which IOC provides a PV

Tags: beginner user developer

This process is for IOCs running on Linux servers.

1.24.1 Find Host and TCP port

The `cainfo` command will tell you which host is serving a particular PV, and which TCP port number on that host is used.

```
$ cainfo LN-TS{EVR:1A-SFP}Pwr:RX-I
LN-TS{EVR:1A-SFP}Pwr:RX-I
State:            connected
Host:             10.0.152.111:5064
Access:           read, write
Native data type: DBF_DOUBLE
Request type:     DBR_DOUBLE
Element count:    1
```

Here we see that the PV "LN-TS{EVR:1A-SFP}Pwr:RX-I" is served from port number 5064 of 10.0.152.111.

```
$ cainfo LN-RF{AMP:1}Amp-Sts
LN-RF{AMP:1}Amp-Sts
State:            connected
Host:             linacioc01.cs.nsls2.local:36349
Access:           read, write
Native data type: DBF_ENUM
Request type:     DBR_ENUM
Element count:    1
```

Here is another example where the hostname is shown instead of an IP address. Also this server has more than one IOC, and the one in question is using port 36349.

1.24.2 Find which process is using a TCP port (Linux only)

Super-user (root) permission is required to find which Linux process is bound to a particular TCP port.

To continue the example from above. On the server linacioc01.cs.nsls2.local we run:

```
$ sudo netstat -tlpn | grep 36349
tcp        0      0 0.0.0.0:36349           0.0.0.0:*               LISTEN      4627/
↪s7ioc
```

This tells us that TCP port 36349 is bound by process ID (PID) 4627, which has the process name of 's7ioc'.

1.24.3 Find information about a process (Linux only)

The `ps` command can give some information, including the command used to start the process. This often contains enough information to identify where the IOC's files can be found.

```
$ ps aux|grep 4627
softioc   4627  1.5  0.0  93748  6616 pts/23   Ssl+ Jan07 744:18 ../../bin/linux-x86/
↪s7ioc /epics/iocs/RF-CONTROL/iocBoot/iocrf-control/st.cmd
```

There are several pieces of information available under `/proc` which are useful. The entry `/proc/<pid>/cwd` is a symbolic link to the current working directory of the process. There is also `/proc/<pid>/exe` which links to the executable.

```
$ sudo ls -l /proc/4627/cwd
lrwxrwxrwx 1 softioc softioc 0 Feb 10 11:49 /proc/4627/cwd -> /epics/iocs/RF-CONTROL
$ sudo ls -l /proc/4627/exe
lrwxrwxrwx 1 softioc softioc 0 Jan  7 09:58 /proc/4627/exe -> /epics/iocs/RF-CONTROL/bin/
↪linux-x86/s7ioc
```

1.24.4 Additional: Finding the procServ/screen running an IOC (Linux only)

The `ps` command can also tell us the PID of the parent of the IOC process. The techniques of step 3 can also be applied to the parent.

```
$ ps -eo pid,ppid,user,cmd|grep 4627
4627  4566 softioc  ../../bin/linux-x86/s7ioc /epics/iocs/RF-CONTROL/iocBoot/iocrf-
↪control/st.cmd
```

The parent PID in the second column is 4566.

```
$ ps aux|grep 4566
softioc   4566  0.0  0.0   3452   592 ?        Ss   Jan07   2:18 /usr/bin/procServ -q -c␣
↪/epics/iocs/RF-CONTROL/iocBoot/iocrf-control -i ^D^C^] -p /var/run/softioc-RF-CONTROL.
↪pid -n RF-CONTROL --restrict --logfile=/var/log/softioc-RF-CONTROL.log 4057 /epics/
↪iocs/RF-CONTROL/iocBoot/iocrf-control/st.cmd
```

And to complete the circle, and get access to the IOC console, we find which TCP port this procServ instance is bound to.

```
$ sudo netstat -tlpn|grep 4566
tcp        0      0 127.0.0.1:4057          0.0.0.0:*               LISTEN      4566/
↪procServ
$ telnet localhost 4057
epics> dbpr LN-RF{AMP:1}Amp-Sts
ASG:                DESC: Ampl.500 MHz E-Source             DISA: 0
DISP: 0             DISV: 1             NAME: LN-RF{AMP:1}Amp-Sts
RVAL: 16            SEVR: NO_ALARM      STAT: NO_ALARM      SVAL: 0
TPRO: 0             VAL: 1
```

1.25 How to Make Channel Access Reach Multiple Soft IOCs on a Linux Host

Tags: developer advanced

1.25.1 UDP Name Resolution: Broadcast vs. Unicast

Running multiple IOCs on one host has an annoying side effect: Clients that are using that host's IP address in their `EPICS_CA_ADDR_LIST` with `EPICS_CA_AUTO_ADDR_LIST=NO` will only reach one of the IOCs – usually the one that was started last. All clients have to use broadcasts to reach all IOCs.

The same is true for CA Gateway machines that are set up in a way that makes multiple Gateway processes serve channels into the same network.

The reason is that the kernel delivers UDP broadcasts to *all* processes that are listening to the IP port, while UDP unicast messages will only be delivered to *one* of those processes.

1.25.2 Fix Using iptables

Here's a little helper (for Linux hosts) that I recently was playing around with – based on an idea by Rodrigo Bongers (CNPEM, Brazil).

If you drop the right script in the right place (depending on your Linux distribution, see further down), it will automatically create/delete an iptables rule that replaces the destination address of all incoming CA UDP traffic on each interface with the broadcast address of that interface.

A simple and effective trick: the kernel will see all incoming name resolution requests as broadcasts, and delivers them to all IOCs instead of one.

Note: This will not work for clients on the same host. (Adding that feature makes things a lot more complicated, and I like things to be simple.)

If you need connections between IOCs on one host, I would suggest adding the broadcast address of the loopback interface (usually `127.255.255.255`) to each IOC's `EPICS_CA_ADDR_LIST` setting.

On Debian and Derivatives

Drop/link the following script into `/etc/network/if-up.d/` and `/etc/network/if-down.d/`. If your system does not have the ip command, consider updating it (or install package iproute2 from backports).

```
#!/bin/sh -e
# Called when an interface goes up / down

# Author: Ralph Lange <Ralph.Lange@gmx.de>

# Make any incoming Channel Access name resolution queries go to the broadcast address
# (to hit all IOCs on this host)

# Change this if you run CA on a non-standard port
PORT=5064

[ "$METHOD" != "none" ] || exit 0
[ "$IFACE" != "lo" ] || exit 0
```

(continues on next page)

(continued from previous page)

```
line=`ip addr show $IFACE`
addr=`echo $line | grep -Po 'inet \K[\d.]+'`
bcast=`echo $line | grep -Po 'brd \K[\d.]+'`
[ -z "$addr" -o -z "$bcast" ] && return 1

if [ "$MODE" = "start" ]
then
    iptables -t nat -A PREROUTING -d $addr -p udp --dport $PORT -j DNAT --to-destination
↪$bcast
elif [ "$MODE" = "stop" ]
then
    iptables -t nat -D PREROUTING -d $addr -p udp --dport $PORT -j DNAT --to-destination
↪$bcast
fi

exit 0
```

On RedHat and Derivatives

On systems using NetworkManager, drop the script below into `/etc/NetworkManager/dispatcher.d/`. The rules for these files are:

> NetworkManager will execute scripts in the /etc/NetworkManager/dispatcher.d directory in alphabetical order in response to network events. Each script should be (a) a regular file, (b) owned by root, (c) not writable by group or other, (d) not set-uid, (e) and executable by the owner. Each script receives two arguments, the first being the interface name of the device just activated, and second an action.

```
#!/bin/sh -e
# Called when an interface goes up / down

# Author: Ralph Lange <Ralph.Lange@gmx.de>

# Make any incoming Channel Access name resolution queries go to the broadcast address
# (to hit all IOCs on this host)

# Change this if you run CA on a non-standard port
PORT=5064

IFACE=$1
MODE=$2

[ "$IFACE" != "lo" ] || exit 0

line=`/sbin/ip addr show $IFACE`
addr=`echo $line | grep -Po 'inet \K[\d.]+'`
bcast=`echo $line | grep -Po 'brd \K[\d.]+'`

[ -z "$addr" -o -z "$bcast" ] && return 1

if [ "$MODE" = "up" ]
```

(continues on next page)

(continued from previous page)

```
then
    /sbin/iptables -t nat -A PREROUTING -d $addr -p udp --dport $PORT -j DNAT --to-
↪destination $bcast
elif [ "$MODE" = "down" ]
then
    /sbin/iptables -t nat -D PREROUTING -d $addr -p udp --dport $PORT -j DNAT --to-
↪destination $bcast
fi

exit 0
```

Enjoy!
Ralph

1.26 How to Set Up a Soft IOC Framework on Linux

This evolved from my notes when doing this for the soft IOCs at BESSY. Please add or correct things as you find them wrong or out-of-date.

The following instructions are based on our Debian Linux machines. (Which version? I don't really care. *Too stable*, I guess.) Other distributions (or other Unixes) might have different commands and different places for things. This is especially true for the Debian `/etc/init.d` script I'm attaching to this page. If you create a different script for a different distribution, please add it to this page. Others will be able to use it. The general steps will be the same on all distributions, though.

Knowledge of general system administration tasks (creating user accounts etc.) is assumed.

I was giving a talk on Administration of Soft IOCs under Linux at the EPICS Collaboration Meeting in April 2007 that partly covered this issue.

1.26.1 Introduction

Why are we doing this?

When using soft IOCs in production, they should be treated as important system services:

- Soft IOCs should be started and stopped by the system.
- There should be a fallback system you can easily switch over to in case of hardware failures.

Other objectives were: In the same way as for VME IOCs, the application developer should be able to reset the soft IOC without needing root access to the host.

- The IOC application developer should be able to start and stop IOCs manually.

When multiple soft IOCs share the same host (and the same IP address), Channel Access can not tell them apart. Access Security will not be able to distiguish between CA connections coming from different soft IOCs. When debugging CA clients, CA will not be able to tell you which of the soft IOCs a connection goes to.

- Channel Access should be able to distinguish between different soft IOCs, even if they are hosted on the same machine.

I was considering using a virtualization layer (based on VMware) to allow running soft IOCs in an encapsulated environment. I found the effort too high, the layer too thick, and the expected performance hit too hard – only to get a separate IP address for each soft IOC.

When debugging and/or trying to look what is happening on an IOC, the developer does not necessarily know if the database is running on a VME based or on a host based soft IOC.

- Console access (and logging console output) should be uniform: working the same way for soft as for VME IOCs.

The setup necessary to achieve this is described in the document *How to Set Up Console Access and Logging for VME and Soft IOCs*.

Concept

To allow Access Security telling the soft IOCs apart, they are run under separate user names.

The procServ utility will be used as an environment that allows to start soft IOCs in the background and connect to their consoles later, much like the serial consoles of VME IOCs. (See the procServ link on the Extensions Page. Formerly, the screen facility was used, but reported problems, e.g. IOCs hanging up after console access, made us change to something less complex.)

Attaching to a soft IOC console will be done through ssh, using a special console access key. Ssh is set up with the matching telnet commands that reattach to the soft IOCs. Opening an ssh connection using the console access key to the user ioc123 on the soft IOC host will immediately attach to the console of the soft IOC named ioc123 (that is running as user ioc123).

1.26.2 Setting up Your Machine

Create User Accounts and ssh Access

Soft IOC Administrator Account

Create a generic user account that application developers will use to start/stop soft IOCs. (We call it iocadm.)

Put the public ssh keys of the application developers into `~/.ssh/authorized_keys` of iocadm.

ssh Key Pairs

As iocadm, create one key pair for this user, and another key pair for console access.

Soft IOCs

Create one user account for each soft IOC you intend to host. User name should be the IOC name, the group is not really important. (Maybe create a group iocs that you put all of them into?)

Into each of the `~/.ssh/authorized_keys` files, put two public keys:

1. The public ssh key of iocadm.
2. The public ssh key for console access.

In front of the console access key, put the telnet command to reattach to the soft IOC console. For a user/IOC ioc123 that provides console access on port 24703, the line should look like this:

```
command="telnet localhost 24703" ssh-rsa AAAAB3NzaC1yc2EAAAA.....
```

Configure the sudo Facility

Allow the iocadm User to Start and Stop Soft IOCs

On the soft IOC host, allow iocadm to use sudo to execute commands as any of the soft IOC users. `/etc/sudoers` should have a line like:

```
iocadm ALL = (ioc123, ioc124, ioc125) NOPASSWD: ALL
```

Setup the Start/Stop script

Create the /etc/init.d script

I'm attaching the script that we're using as `/etc/init.d/softIOC`. It got quite huge and complex – *sorry!*. It has been modelled after Debian's skeleton scripts, you should probably adapt it to match the standards that your distribution implies.

It contains the local settings for where to find things, routines to read in the configuration file, the code necessary to start/stop a soft IOC as a different user under procServ, and the usual init.d script stuff that checks command line arguments and calls the other routines.

If you have a script working for a different distribution, please add it to this page, as it could make life easier for others!

Create the Configuration File

The configuration file contains a section for each of the soft IOCs. A section starts with the IOC name followed by a colon, and ends with an empty line.

Within a section you can set special variables used by the softIOC script as well as environment variables that will be set for the soft IOC.

The special section global: contains settings that will be applied to *all* soft IOCs (may be overridden by the IOC section).

The special line auto: contains the names of the soft IOCs that should be started when the script is run as part of the system boot-up process.

Section and IOC names are not case sensitive.

So a minimal configuration file could look like this (remember the empty line that is required after each section):

```
AUTO:   ioc123

GLOBAL:

ioc123:

ioc124:

ioc125:

```

Distribute the Required Stuff to the Soft IOC Host

EPICS Base

Soft IOCS will need libraries from EPICS base. Make sure these are existing and can be found.

Code and Databases

Add the soft IOC host to the code deployment scheme you are using. The soft IOC binaries, databases, and start up scripts must be available for the soft IOCs to be started.

1.26.3 Start Your Soft IOCs

Start the IOCs using the startup script

Starting and stopping the soft IOCs should work now! Ssh to the soft IOC host as iocadm and try calling the startup script:

```
/etc/init.d/softIOC start ioc123
```

Watch them run

Ssh to the soft IOC host using the console access key and see if you can get access to the IOC console:

```
ssh -i ~iocadm/.ssh/console_access -t ioc123@iochost
```

You should be directly connected to your IOC's console.

Check if Starting IOCs at reboot works

If you made entries to the auto: section, reboot the machine to check that starting IOCs at boot time works.

Good luck!

Ralph Lange (BESSY)

1.26.4 The Startup Script

`/etc/init.d/softIOC` script for Debian Linux

```
#! /bin/sh
# Author: <Ralph.Lange@bessy.de>
#
# History:
#          2006-03-12: Adapted from D. HerrendÃ¶rfer's ca-gateway script
#          2006-04-04: Bugfix in config file parser
#          2008-05-20: Adapted to procServ
```

(continues on next page)

(continued from previous page)

```
# Do NOT "set -e"

# !! This script is located on a mounted file system
# !! It must be run after the mountnfs.sh script

PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="EPICS soft IOCs"
SCRIPTNAME=/etc/init.d/softIOC
HOST=`uname -n`


PROCSERV=/usr/local/bin/procServ
CONFFILE=/opt/IOC/softIOC/softiocs.$HOST
HOMEDIRS=/home/controls


# Check for config file
if [ ! -r $CONFFILE ]
then
        echo "Error: Can't find configuration file $CONFFILE!"
        exit 1
fi

#
# Functions that read in the configuration file
#
clear_options()
{
        for option in "CA_AUTO" "CA_ADDR" "CA_PORT" "IOC_USER" "PORT"
        do
                unset $option;
        done
}

evaluate_options()
{
        while [ $# != 0 ]
        do
                TAG=`echo $1 | tr [:lower:] [:upper:]`
                case "$TAG" in
                "#")            ;;
                "CA_AUTO" | "CA_ADDR" | "CA_PORT" | "COREDUMPSIZE" | \
                "HOMEDIR" | "BOOTDIR" | "IOC_USER" | "PORT" )
                                # Test the presence of values for the current option
                                OPTION=$TAG
                                shift
                                if [ -z $TAG -o $TAG = "#" ]
                                then
                                        echo "$CONFFILE: Value(s) required for $TAG.";
                                        exit 1
                                else
                                        VALUE=$1
```

(continues on next page)

(continued from previous page)

```
                                        shift
                                fi
                                # If more values follow assign them too
                                while [ $1 != '#'  -a  $# != 0 ]
                                do
                                        VALUE="$VALUE $1"
                                        shift;
                                done
                                eval ${OPTION}=\$VALUE
                                ;;
                *)              echo "$CONFFILE: Unknown option $1."
                                exit 1
                esac
                shift
        done
}

default_options()
{
        # Set IOC defaults for options
        # (may be overridden in config file)
        IOC_LC=$1
        IOC_UC=`echo $1 | tr [:lower:] [:upper:]`

        if [[ "$IOC_LC" = "mdi"* ]]
            then
            TOP=DiagR3.14.9.0.1-Tornado2.2.1
        elif [[ "$IOC_LC" = *"p" ]]
            then
            TOP=MLS-Controls
        else
            TOP=BII-Controls/base-3-14
        fi

        BOOTDIR=/opt/IOC/$TOP/boot/$IOC_UC
        HOMEDIR=$HOMEDIRS/$IOC_LC
        PIDFILE=$HOMEDIR/$IOC_LC.pid
        ENVFILE=$HOMEDIR/$IOC_LC.env
        IOC_USER=$IOC_LC
}

assign_options()
{
        # Find $TAG section
        # Remove comments
        # Remove leading and trailing whitespace
        # Remove $TAG: tag
        # Join lines ending with a "\"
        # Mark end of option with a "#"
        # Remove unnecessary whitespace
        TAG=$1
        SECTION=`sed -n "/^$TAG:/I,/^[\t ]*$/p" $CONFFILE | \
```

(continues on next page)

(continued from previous page)

```
                sed -n '/^[^#]/p' | \
                sed -e 's/^[ \t]*//' -e 's/[ \t]*$//' \
                    -e "s/$TAG://I" \
                    -e :a -e '/\\\\$/N; s/\\\\\\n//; ta' \
                    -e 's/$/ \#/' \
                    -e 's/[\t ]/ /g'`
        evaluate_options $SECTION
}

get_iocs()
{
        # Get IOCs from command line or AUTO: entry in configuration file
        # Test for matching section in configuration file
        if [ $# = 0 ]
        then
                TEST_LIST=`grep -i '^AUTO:' "$CONFFILE" | cut -d: -f2- | tr [:upper:]␣
↪[:lower:]`
        else
                TEST_LIST="$@"
        fi

        CHECKED_LIST=""
        for IOC in $TEST_LIST
        do
                grep -qi "^$IOC:" $CONFFILE
                if [ $? = 0 ]
                then
                        CHECKED_LIST="$CHECKED_LIST $IOC"
                fi
        done
        echo $CHECKED_LIST
}

set_cmdenvopts()
{
        # Set up the environment setup string

        SETENV="LINES=60 "`test ! -z "$CA_AUTO" && echo "export EPICS_CA_AUTO_ADDR_LIST=\
↪"$CA_AUTO\";"`
        SETENV="$SETENV "`test ! -z "$CA_ADDR" && echo "export EPICS_CA_ADDR_LIST=\"$CA_
↪ADDR\";"`
        SETENV="$SETENV "`test ! -z "$CA_PORT" && echo "export EPICS_CA_SERVER_PORT=\"
↪$CA_PORT\";"`
        SETENV="$SETENV "`test ! -z "$BOOTDIR" && echo "export BOOTDIR=\"$BOOTDIR\";"`

        # Set up the options for the procserv program

        PROCSERVOPTS=`test ! -z "$IOC_USER" && echo "-n \"$IOC_USER\""`
        PROCSERVOPTS="$PROCSERVOPTS "`test ! -z "$COREDUMPSIZE" && echo "--coresize \"
↪$COREDUMPSIZE\""`
        PROCSERVOPTS="$PROCSERVOPTS -q -c $BOOTDIR -p $PIDFILE -i ^D^C^] $PORT"
}
```

(continues on next page)

(continued from previous page)

```
#
# Function that starts the daemon/service
#
do_start()
{
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started
        # Add code here, if necessary, that waits for the process to be ready
        # to handle requests from services started subsequently which depend
        # on this one.  As a last resort, sleep for some time.

        echo -n "Starting soft IOCs ... "
        MYIOCS=`get_iocs $@`
        [ "$MYIOCS" = "" ] && echo -n "<none> "
        for IOC in $MYIOCS
        do
                echo -n "$IOC "
                clear_options
                default_options "$IOC"
                assign_options "GLOBAL"
                assign_options "$IOC"
                set_cmdenvopts

                if [ -d $BOOTDIR ]
                    then
                    if [ -d $HOMEDIR ]
                        then

                        sudo -H -u $IOC sh -c "$SETENV (env > $ENVFILE; /sbin/start-stop-
↪daemon --start --quiet --chdir $BOOTDIR \
                            --pidfile $PIDFILE --startas $PROCSERV --name procServ --
↪test > /dev/null)"
                        if [ "$?" = 1 ]
                            then
                            echo -n "<was running> "
                            else
                            sudo -H -u $IOC sh -c "$SETENV (env > $ENVFILE; /sbin/start-
↪stop-daemon --start --quiet --chdir $BOOTDIR \
                                --pidfile $PIDFILE --startas $PROCSERV --name procServ --
↪ $PROCSERVOPTS ./st.cmd)"
                            if [ "$?" = 1 ]
                                then
                                echo -n "<failed> "
                            fi
                        fi

                    else
                        echo -e "\nWarning: Home directory $HOMEDIR does not exist!␣
↪Ignoring $IOC"
```

(continues on next page)

(continued from previous page)

```
                    fi
                else
                        echo -e "\nWarning: Boot directory $BOOTDIR does not exist!␣
↪Ignoring $IOC"
                fi
        done
        echo "... done."
}

#
# Function that stops the daemon/service
#
do_stop()
{
        # Return
        #   0 if daemon has been stopped
        #   1 if daemon was already stopped
        #   2 if daemon could not be stopped
        #   other if a failure occurred

        echo -n "Stopping soft IOCs ... "
        MYIOCS=`get_iocs $@`
        [ "$MYIOCS" = "" ] && echo -n "<none> "
        for IOC in $MYIOCS
        do
                echo -n "$IOC "
                clear_options
                default_options "$IOC"
                assign_options "GLOBAL"
                assign_options "$IOC"
                set_cmdenvopts

                sudo -H -u $IOC sh -c "/sbin/start-stop-daemon --stop --quiet --pidfile
↪$PIDFILE --name procServ --test > /dev/null"
                if [ $? = 1 ]
                    then
                    echo -n "<not running> "
                    else
                    sudo -H -u $IOC sh -c "/sbin/start-stop-daemon --stop --quiet --
↪retry=TERM/30/KILL/5 --pidfile $PIDFILE --name procServ"
                    if [ $? = 1 ]
                        then
                        echo -n "<failed> "
                    else
                        sudo -H -u $IOC sh -c "rm -f $PIDFILE"
                    fi
                fi
        done
        echo "... done."
}

#
```

(continues on next page)

(continued from previous page)

```
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
        #
        # If the daemon can reload its configuration without
        # restarting (for example, when it is sent a SIGHUP),
        # then implement that here.
        #
#       start-stop-daemon --stop --signal 1 --quiet --pidfile $PIDFILE --name $NAME
#       return 0

        echo "Restarting soft IOCs ... "
        STARTDIR=$PWD
        IOCS=`get_iocs $@`
        [ "$IOCS" = "" ] && echo -n "<none> "
        for IOC in $IOCS
        do
                echo -n "$IOC "
                clear_options
                default_options "$IOC"
                assign_options "GLOBAL"
                assign_options "$IOC"
                if [ -d $BOOTDIR ]
                then
                        cd "$BOOTDIR"
# restart it!
                        echo -e "\ndebug: Reloading ioc $IOC"
                        cd "$STARTDIR"
                else
                        echo -e "\nWarning: Boot directory $BOOTDIR does not exist!␣
↪Entry for $NET ignored!"
                fi
        done
        echo "... done."
}


COMMAND=$1
shift
IOCS=`echo $@ | tr [:upper:] [:lower:]`

case "$COMMAND" in
  start)
        do_start $IOCS
        ;;
  stop)
        do_stop $IOCS
        ;;
  #reload|force-reload)
        #
        # If do_reload() is not implemented then leave this commented out
        # and leave 'force-reload' as an alias for 'restart'.
```

(continues on next page)

(continued from previous page)

```
        #
        #log_daemon_msg "Reloading $DESC" "$NAME"
        #do_reload
        #log_end_msg $?
        #;;
  restart|force-reload)
        #
        # If the "reload" option is implemented then remove the
        # 'force-reload' alias
        #
        do_stop $IOCS
        sleep 1
        do_start $IOCS
        ;;
  *)
        #echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}" >&2
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload} [iocs ...]" >&2
        exit 3
        ;;
esac
```

1.27 How to Set Up Console Access and Logging for VME and Soft IOCs

This evolved from my notes when doing this for the IOCs at BESSY. Please add or correct things as you find them wrong or out-of-date.

The following instructions are based on our Debian Linux machines. (Which version? I don't remember. *Too stable*, I guess.) Other distributions (or other Unixes) might have slightly different commands and different places for things. The general steps will be the same on all distributions, though.

Knowledge of general system administration tasks (creating user accounts, using ssh, rsync etc.) is assumed.

I was giving a talk on Administration of Soft IOCs under Linux at the EPICS Collaboration Meeting in April 2007 that partly covered this issue.

1.27.1 Introduction

Why are we doing this?

When debugging and/or trying to look what is happening on an IOC, the developer does not necessarily know if the database is running on a VME based or on a host based soft IOC.

- Console access (and logging console output) should be uniform: working the same way for soft as for VME IOCs.
- Connecting to a console should be easy. It should not require intimate knowledge of the system structure, the name of the IOC should be all that is needed for connecting to its console.
- Multiple users should be able to log onto the same console. Only one of those should be granted write access. Forcing to take over write access must be possible, so that no one is able to block a console by not logging off.
- Viewing log files should be as easy as pointing your browser to a certain URL.

Conserver

Conserver is a free software that does provide all of the functionality needed plus a lot more things.

From its docs:

> Conserver is an application that allows multiple users to watch a serial console at the same time. It can log the data, allows users to take write-access of a console (one at a time), and has a variety of bells and whistles to accentuate that basic functionality. The idea is that conserver will log all your serial traffic so you can go back and review why something crashed, look at changes (if done on the console), or tie the console logs into a monitoring system (just watch the logfiles it creates). With multi-user capabilities you can work on equipment with others, mentor, train, etc. It also does all that client-server stuff so that, assuming you have a network connection, you can interact with any of the equipment from home or wherever.

I will describe a setup consisting of multiple conserver hosts, connect to a set of VME based IOCs (through telnet to terminal servers) and soft IOCs (through ssh). Setting up your soft IOCs to be accessed through ssh is described in the document *How to Set Up a Soft IOC Framework on Linux*.

Multiple Server Setup

Conserver supports running multiple servers, that are aware of the lists of consoles each of the member serves. A conserver-client can ask any of the server nodes for a console, the server will automatically redirect the request to the appropriate server node.

Main advantage: client machines never need any re-configuration, if consoles are moved between servers or additional consoles are added to the system. They have to know the name of console they want to attach to and one of the servers (preferably through a DNS alias name) - that's it.

1.27.2 Setting up Your Machines

Get conserver

The machines intended to run conserver (i.e. the nodes that connect to consoles, provide console access, and log console output), need the conserver-server package installed.

The machines intended to be clients (i.e. the nodes where the console application can be run), need the conserver-client package installed.

Configure the Conserver Servers

Conserver's configuration files use a straightforward configuration languange with lots of different options and very few structures. (See `man conserver.cf` for more details.)

It's a bit like configuring bootp or dhcp: The most important structure is a define command for aliases, that allow inclusion of other defines, adding or overriding parameters. Using a smart set of those definitions, the actual entries for the consoles can be quite short.

Shared vs. Local Configuration

To allow multiple servers to share the same configuration, the `#include` directive is used to add by-host configuration. For each of the servers, a separate configuration file is created for keeping the local configuration. The matching local configuration file is soft-linked to the generic name conserver.local. The shared conserver.cf file includes this file to read the by-host information.

So, e.g. on a conserver server s1, the directory `/etc/conserver` looks like

```
-rw-r----- 1 root     root 28785 Apr 18 12:15 conserver.cf
-rw-r----- 1 root     root   201 Mar 30 15:48 conserver.s1.cf
-rw-r----- 1 root     root   205 Mar 30 15:48 conserver.s2.cf
lrwxrwxrwx 1 root     root    20 Mar 30 15:29 conserver.local.cf -> conserver.s1.cf
```

That way, you can distribute a complete set of configuration files to multiple servers without damaging the setup.

The Local Configuration File

Local configuration will usually only set the master for certain console groups, i.e. it enables the server to decide which console lines it serves itself, and to which other master it should redirect requsts for other console lines.

For a server that should host the consoles for the project "bii" locally, and redirect the requests for "mls" consoles to another server, the local configuration looks like this:

```
default m-mls { master s2.mls.bessy.de; }
default m-bii { master localhost; }
```

Users and Groups

Set up a scheme of users and groups that you want to use. Authentification can be done by a password file or using PAM. Users and group definitions are used to grant access rights (read-only, read-write, admin) to certain users and groups. Decide what you need, and configure it.

We are using something like

```
group grp-adm  { users me, you, doubleyou; }

group grp-tsc  { users grp-adm, tscadm; }
group grp-id   { users grp-adm, idadm; }
```

to have people from two different areas (controls, insertion devices) use a password for write access to their IOCs.

Access to the Server

Define who will be the administrators on the server and which networks you will be allowing access from.

In this case: Use the group of administrators defined above, and allow access from all local (private) IPs.

```
access * {
        admin   grp-adm;
        allowed 127.0.0.1, 192.168.0.0/16;
}
```

Defaults for the Server

This block provides a reasonable set of default definitions for things. Some may even not be necessary, they're just here for clarification.

```
config localhost {
        autocomplete true;
        defaultaccess rejected;
        initdelay 60;
        logfile /var/log/conserver/conserver/conserver.log;
        passwdfile /etc/conserver/conserver.passwd;
        primaryport 782;
        redirect yes;
        reinitcheck 1;
        secondaryport 0;
        sslrequired false;
        unifiedlog /var/log/conserver/unified.log;
}
```

Defaults for the Groups

In this case, just make the logs go in different subdirectories, and allow read/write access for the groups defined above.

```
default tsc-def  { logfile /var/log/conserver/tsc/&.log;   rw grp-tsc; }
default id-def   { logfile /var/log/conserver/id/&.log;    rw grp-id; }
```

Message of the Day and Time Stamping

This defines what connecting users will see as a warning when they connect.

```
default message {
        motd \\
                                WARNING!!!
This is only an example! ; }
```

General default that includes the message just defined and adds the definition of time stamps that conserver adds to log entries: at every line.

```
default def {
        include message;
        timestamp "1lb";
}
```

Include the Local Definitions

Include the local by-host configuration (as mentioned above).

```
#include /etc/conserver/conserver.local.cf
```

Defining the Soft IOC Hosts

Here the host machines for the soft IOCs are defined. (So that the conserver knows where to ssh to.)

```
default h-mls-sioc { host iochost.mls.bessy.de; }
default h-bii-sioc { host iochost.bii.bessy.de; }
```

Defaults for Console Types

These are the generic definitions for soft IOCs and terminal server based IOCs for the different groups and projects. We do it in a way so that the instances of console lines will only have to include one definition and add the stuff needed for that instance. the examples show the definitions for soft IOCs and terminal servers (telnet access to port 2001...2016 for a 16 port terminal server). The first include is a definition from the local configuration file, telling the server which of the servers that console is connected to (who is the master of the console). The other included defintions were shown above.

```
default bii-sioc {
        include m-bii;
        include def;
        include tsc-def;
        include h-bii-sioc;
        type exec;
        execrunas iocadm;
        exec /usr/bin/ssh -i /home/controls/iocadm/.ssh/conserver -t U@H;
        execsubst U=cs,H=hs;
}
```

```
default bii-ts {
        include m-bii;
        include def;
        include tsc-def;
        type host;
        portbase 2000;
}
```

Lists of Terminal Servers

Next are the lists of the existing terminal servers, that VME IOC consoles (or other hardware) are connected to.

```
default ts2    { include bii-ts; host ts2.bii.bessy.de; }
default ts3    { include bii-ts; host ts3.bii.bessy.de; }
```

Lists of Console Instances

Finally, there are the lists of console instances. One line per each soft IOC or VME IOC that we are connecting to. Have a look at two VME IOCs on terminal server 2, one on terminal server 3, plus three soft IOCs.

```
console ioc1    { include ts2; port  1; include tsc-def; aliases ts2-01; }
console ioc2    { include ts2; port  2; include tsc-def; aliases ts2-02; }
console ioc3    { include ts3; port  1; include tsc-def; aliases ts3-01; }
```

```
console sioc1   { include bii-sioc; }
console sioc2   { include bii-sioc; }
console sioc3   { include bii-sioc; }
```

Start the Conserver Servers

`/etc/init.d/conserver-server start` (or the equivalent on your Linux distribution) should get things running.

Entering it in your system configuration will make sure it gets started when the machine boots.

Caveat: Memory Leak

There is a memory leak in the conserver server (as of version 8.1.14). We have set up a cron job that restarts the conserver server once a week to get around. This bug is known to the conserver developers - hopefully it will be fixed some time soon.

Configure the Conserver Clients

After installing the conserver-client package, the system-wide configuration file `/etc/console.cf` will contain the default configuration for the console clients.

It might be a good idea to add a DNS alias console to all your networks, so that all console configurations can point to the same master. Even if you move a conserver to a different machine, you will be able to change the DNS alias and don't have to reconfigure all clients.

1.27.3 Set up Web Browsing for the Log Files

Collect the Log Files

On your web server, create a location where the conserver logs are to be placed. Set up cron jobs that collect the logfiles from your servers using commands like this:

```
/usr/bin/rsync -a --delete consync@s1.bii.bessy.de:/var/log/conserver /web/conserver-bii␣
↪>> /web/conserver-bii/consync.log 2>&1
```

(This one requires a user account for consync being created on the conserver server s1, and that the collecting user account having ssh access to it.)

Setup the Web Server

This is well outside the scope of this Wiki page and should be fairly easy - refer to the documentation of your web server to learn about it.

Good luck!

Ralph Lange (BESSY)

1.28 PV Save and Restore Tools available

There are a number of different tools available within the EPICS community for saving and restoring values of PVs. This page gives somewhere for them to be briefly described, since they all have somewhat different characteristics. This page may be incomplete; if you know of a published tool that is not described here, please consider adding it

1.28.1 IOC-based Tools

1.28.2 SynApps Autosave

Autosave automatically saves the values of EPICS process variables (PVs) to files on a server, and restores those values when the IOC is rebooted.

1.28.3 Host-based Tools

XAL 'score'

The XAL collection of high-level applications includes 'score': A Table of PV names, each with a current value and a saved value. One can see which PVs differ from their saved state, restore them etc.

Don't know a good web page, but try this one .

CSS 'PV Table'

Similar to 'score', but simpler and still unter development. Saves values to XML files, no connection to RDB.

For more on CSS (Control System Studio), see the main page at DESY.

sddscasr

sddscasr is based off of casave, carestore, and sddssnapshot. It is used to save and restore snapshots of the PV values. The format of the request file and snapshot file is SDDS. It can be run in daemon mode where it maintains connections to PVs and when a trigger PV is activated it creates a new snapshot. This reduces the number of network calls to all the IOCs because it does not have to search for the PVs every time a snapshot it taken. This program is used at the APS for many different systems, including some with request files having over 17 thousand PVs.

It can also be used in conjunction with APS's SaveCompareRestore tool.

1.29 Channel Access Protocol Specification

Tags: advanced

Table of Contents

- *Example message*
- *Repeater Operation*
  - *Startup*
  - *Client detection*
  - *Operation*
  - *Shutdown*
- *Searching Strategy*
- *ECA Error/Status Codes*
- *Example conversation*
- *Glossary of Terms*
- *References*

1.29.1 License

1.29.2 Document History

Revision	Date	Author	Section	Modification
1.0	2003-12-12	Klemen Žagar	all	Created.
1.1	2004-01-08	Aleš Pucelj	all	Finalized structure.
	2004-01-10	Matej Šekoranja	all	Review.
1.2	2004-04-19	Aleš Pucelj	all	Draft completed.
1.3	2004-05-31	Aleš Pucelj	all	Matej's comments considered (after Channel Access for Java implementation).
	2004-06-01	Matej Šekoranja	all	Review.
	2004-08-12	Klemen Žagar	all	Released
1.4	2008-02-07	Matej Šekoranja	all	Description of `CA_PROTO_READ` and `CA_PROTO_READ_SYNC` added.
	2008-02-07	Klemen Žagar	all	Released
1.4.1	2014-08-27	Daniel J. Lauk	all	Transformed to AsciiDoc format. Recreated graphics.
1.5	2014-09	Michael Davidsaver	all	Major revision to describe operation semantics
1.6	2019-09-05	Ian Gillingham	all	Minor revision migrated to Readthedocs via Shpinx build from rst source

1.29.3 Introduction

This document describes the EPICS Channel Access (CA) protocol as it is, and has been, implemented. It is also intended to act as a specification to allow the creation of new client and server implementations. The focus is on versions >= 4.11 of the CA protocol, which is used by EPICS Base 3.14.0 and later. No changes from protocol versions before 4.8 (EPICS Base 3.13.0) will be included in this document.

For the benefit of those writing new clients and servers RFC 2119:Key words for use in RFCs to Indicate Requirement Levels are used.

1.29.4 Concepts

Process Variables

A Process Variable (PV) is the addressable unit of data accessible through the Channel Access protocol. Each PV has a unique name string and SHOULD be served by a single Channel Access server. Specifically, when searching for a PV, each client MUST NOT receive replies identifying more than one server.

Virtual Circuit

A TCP connection between a CA client and server is referred to as a Virtual Circuit.

Typically only one Circuit is opened between each client and server. However, a client MAY open more than one Circuit to the same server.

TCP Message Flow

The following tree diagram illustrates the order in which normal (not error) CA messages can be sent on a TCP connection. Nodes with box borders are messages sent be the server, and oval borders are messages sent be the client. Nodes with a double border (eg. "Open Socket") are not themselves messages. Instead they indicate pre-conditions which must be meet before certain messages can be sent.

The message CA_ERROR may be sent by a server in response to any client message.

Channels

A Channel is the association between a particular Circuit and PV name.

At core, a Channel is a runtime allocated pair of integer identifiers (CID and SID) used in place of the PV name to avoid the overhead of string operations. Both client and server MUST maintain a list of the identifiers of all open Channels associated with a Circuit.

The scope of these identifiers is a single Circuit. Identifiers from one Circuit MUST NOT be used on any other. Further more, the same identifier number may be used one two different Circuit in connection with two different PV names.

A Channel's identifiers are explained in section *Message Identifiers*.

Monitors

A monitor is created on a channel as a means of registering/subscribing for asynchronous change notifications (publications). Monitors may be filtered to receive only a subset of events (Event Mask), such as value or alarm changes. Several different monitors may be created for each channel.

Clients SHOULD NOT create two monitors on the same channel with the same Event Mask.

Server Beacons

Server beacons messages (*CA_PROTO_RSRV_IS_UP*) MUST be periodically broadcast. Beacon messages contain the IP address and TCP port on which the server listens A sequential beacon ID is also included.

When a server becomes active, it MUST immediately begin sending beacons with an increasing delay. An initial beacon interval of 0.02 seconds is RECOMMENDED. After each beacon is sent the interval SHOULD be increased up to a maximum interval. Doubling the interval is RECOMMENDED. The RECOMMENDED maximum interval is 15 seconds.

As a server sends beacons it MUST increment the BeaconID field for each message sent.

CA clients MAY use a server's first beacon as a trigger to re-send previously unanswered *CA_PROTO_SEARCH* messages.

While it was done historically, clients SHOULD NOT use Beacons to make timeout decisions for TCP Circuits. The *CA_PROTO_ECHO* message should be used instead.

Clients wishing to detect new servers should maintain a list of all servers along with the last BeaconID received, and the reception time. Servers SHOULD be removed from this list when no Beacon is received for some time (two beacon periods is RECOMMENDED).

Repeater

See *Repeater Operation*.

Timeout Behavior

CA clients typically SHOULD NOT automatically reconnect Circuits which have become unresponsive, instead CA clients SHOULD send a new *CA_PROTO_SEARCH* request.

CA clients SHOULD on occasion re-send PV name searches which are not answered.

Care must be taken to avoid excessive network load due to repeated lookups and connections. Clients are RECOMMENDED to implement an exponentially increasing (up to a maximum) interval when re-sending CA_PROTO_SEARCH messages for each PV.

Clients are RECOMMENDED to implement a timeout before re-starting a search when a Channel is closed due to an Exception, or Channel creation fails with *CA_PROTO_CREATE_CH_FAIL* reply.

Version compatibility

Certain aspects of Channel Access protocol have changed between releases. In this document, Channel Access versions are identified using `CA_VXYY`, where X represents single-digit major version number and YY represents a single- or double-digit minor version number. Stating that a feature is available in `CA_VXYY` implies that any client supporting version XYY must support the feature. Implementation must be backward compatible with all versions up to and including its declared supported minor version number.

Example 1. Channel Access version number

`CA_V43`, denotes version 4.3 (major version 4, minor version 3).

Channel Access protocol carries an implicit major version of 4. Minor version begin with 1. Minor version 0 is not a valid version.

When a Virtual Circuit is created both client and server send their minor version numbers. The valid messages and semantics of the Circuit are determined by the lower of the two minor versions.

A partial history of CA minor version changes:

EPICS Base	CA Minor	Year	Reason
3.14.12-pre1	13	2010	Dynamic array size in monitors
3.14.12	12	2010	PV search over tcp
3.14.0-b2	11	2002	large array?, circuit priority?
3.14.0-b2	10	2002	Beacon counter???
3.14.0-b1	9	2001	Large packet header
3.13.0-b10	8	1997	??
3.13.0-a5	7	1996	Start of CVS history
3.12.1.3	6	1995	???
3.12.1.1	5	1995	???
3.12.0-beta1	4	1994	???
3.12.0-beta0	3	1994	Start of GIT history

Exceptions

Channel Access protocol error messages (*CA_PROTO_ERROR*) are referred to as Exceptions. Exceptions are sent by a CA server to indicate its failure to process a client message.

An Exception MAY be sent in response to any client message, including those which normally would not result in a reply.

Exception messages carry the header of the client message which triggered the error. It is therefore always possible to associate an Exception with the request which triggered it.

1.29.5 Operation

Overall Server Operation

A CA server will maintain at least two sockets.

A UDP socket bound to the CA port (def. 5064) MUST listen for PV name search request broadcasts. PV name search replies are sent as unicast messages to the source of the broadcast. This socket, or another UDP socket, SHOULD periodically send Beacons to the CA Beacon port (def. 5065).

A TCP socket listening on an arbitrary port. The exact port number is included in PV name search replies. This socket will be used to build Virtual Circuits.

A CA server SHOULD NOT answer PV name search requests for itself unless a *CA_PROTO_CREATE_CHAN* for that PV from the same client can be expected to succeed. To do otherwise risks excessive load in a tight retry loop.

Overall Client Operation

A CA client SHOULD maintain a registration with a Repeater on the local system, (re)starting it as necessary.

Clients will send PV name search messages and listen for replies. Typically a client will maintain a table of unanswered name searches and a cache of recent results in order avoid duplicate searches, and to process any replies.

Once an affirmative search reply is received, a Virtual Circuit to the responder is opened if needed. If the client already has a circuit open to this server, it SHOULD be reused. When a Circuit is available, a Channel is created on it, then various get/put/monitor operations are performed on this Channel.

Name Searching

The process of finding the server which advertises a PV to a particular client can be carried out over UDP, or with >= `CA_V412` over a TCP connection.

In either case each client SHOULD be pre-configured with a set of destinations to send queries. For UDP searching, this is a list of unicast or broadcast endpoints (IP and port). For TCP searching, this is a list of endpoints.

It is RECOMMENDED that a default set of UDP endpoints be populated with the broadcast addresses of all network interfaces except the loopback.

It is RECOMMENDED that, on client startup, Circuits be established to all endpoints in the TCP search list.

Search results are transitory. Subsequent searches MAY yield different results. Therefore queries SHOULD be re-tried unless an active Channel is already open.

UDP search datagrams

Several CA messages MAY be included in one UDP datagram.

A datagram which includes *CA_PROTO_SEARCH* messages MUST begin with a *CA_PROTO_VERSION* message.

For efficiency it is RECOMMENDED to include as many search requests as possible in each datagram, subject to datagram size limits.

A CA server MUST NOT send a *CA_PROTO_NOT_FOUND* in response to a UDP search request.

TCP search

CA_PROTO_SEARCH messages MUST NOT be sent on a Circuit unless a *CA_PROTO_VERSION* message has been received indicating >= `CA_V412`.

When supported, *CA_PROTO_SEARCH* messages may be sent at any time the circuit is open.

A CA server MAY send a *CA_PROTO_NOT_FOUND* in response to a UDP search request if the DO_REPLY bit is set.

Clients MAY ignore *CA_PROTO_NOT_FOUND* messages.

A *CA_PROTO_NOT_FOUND* message is not final. A subsequent search might yield a different result.

Virtual Circuits

Inactivity timeout

When a Circuit is created, both client and server MUST begin a countdown timer. When any traffic (including a *CA_PROTO_ECHO* message) is received on the Circuit, this counter is reset to its initial value. If the timer reaches zero, the Circuit is closed.

Clients MUST send a *CA_PROTO_ECHO* message before the countdown reaches zero. It is RECOMMENDED to send an echo message when the countdown reaches half its initial value.

When a *CA_PROTO_ECHO* message is received by the server, it MUST be immediately copied back to the client.

The RECOMMENDED value for the countdown timer is 30 seconds.

Circuit Setup

When a Circuit is created, both client and server MUST send *CA_PROTO_VERSION* as their first message. This message SHOULD be sent immediately.

Note for implementers. For EPICS Base before 3.14.12, RSRV did not immediately send a version message due to a buffering problem. Instead the version message was not sent until some other reply forced a flush of the send queue.

In addition the client SHOULD send *CA_PROTO_HOST_NAME* and *CA_PROTO_CLIENT_NAME* messages. Once this is done, the Circuit is ready to create channels.

Note that the host and client name messages SHOULD NOT be (re)sent after the first channel is created. If the client or host name strings change, the circuit SHOULD be closed.

If no host or client name messages are received a server MUST consider the client to be anonymous. It is RECOMMENDED that anonymous users not be granted rights for the Put operation.

Channel Creation

Channel creation starts with a *CA_PROTO_CREATE_CHAN* request from the client. This message includes the PV name string, and a client selected *CID*.

If the server can not provide the named PV it replies with *CA_PROTO_CREATE_CH_FAIL* using the same CID. The server MUST NOT remember the CID of failed creation requests as clients MAY re-used them immediately.

If the server can provide the named PV, it replies with *CA_PROTO_ACCESS_RIGHTS* followed by a *CA_PROTO_CREATE_CHAN* reply. Further *CA_PROTO_ACCESS_RIGHTS* messages MAY follow to reflect changes to access permissions.

Note that the *CA_PROTO_CREATE_CHAN* reply includes the Channel's native DBR datatype and the maximum number of elements which can be retrieved/set by a get, put, or monitor operation. These attributes are fixed for the lifetime of the channel.

The reply also contains the server selected *SID* identifier. Together with the CID, these two identifier will be used to refer to the Channel in subsequent operations.

The Channel remains active, and the identifiers valid, until a *CA_PROTO_CLEAR_CHANNEL* request is sent by a client and its reply received, until a *CA_PROTO_SERVER_DISCONN* message is received by a client, or if the circuit (TCP connection) is closed.

After a server sends a CA_PROTO_CLEAR_CHANNEL reply or a CA_PROTO_SERVER_DISCONN message it MAY reuse the SID immediately.

After a client receives a CA_PROTO_CLEAR_CHANNEL reply or a CA_PROTO_SERVER_DISCONN message it MAY reuse the CID immediately.

Therefore after a client sends a CA_PROTO_CLEAR_CHANNEL request, or a sever sends a CA_PROTO_SERVER_DISCONN request, no further messages (including *CA_PROTO_ERROR*) should be sent for the closed channel.

Put Operations

A Operation to write data to a Channel begins with a *CA_PROTO_WRITE* or *CA_PROTO_WRITE_NOTIFY* request. The difference between the two is that CA_PROTO_WRITE_NOTIFY gives a reply on success, while CA_PROTO_WRITE does not.

The CA_PROTO_WRITE SHOULD be used when it is not important that all Put operations are executed. A server SHOULD make best effort to ensure that, when a burst of CA_PROTO_WRITE requests is received, that the last request is processed (others could be dropped).

A CA_PROTO_WRITE_NOTIFY request indicates that the client intends to wait until the request is fulfilled before continuing. A server MUST reply to all CA_PROTO_WRITE_NOTIFY requests. A server SHOULD make best effort to fully process all CA_PROTO_WRITE_NOTIFY requests.

Both request messages include a *SID* to determine which Channel is being operated on.

In addition, a client selected *IOID* is included. This identifier will be included in a CA_PROTO_WRITE_NOTIFY reply, as well as any CA_PROTO_ERROR exception message resulting from a Put request.

Get Operation

The present value of a Channel is queried with a *CA_PROTO_READ_NOTIFY* request.

A server MUST reply to all CA_PROTO_READ_NOTIFY requests. A server SHOULD make best effort to fully process all CA_PROTO_READ_NOTIFY requests.

CA_PROTO_READ_NOTIFY messages include a *SID* to determine which Channel is being operated on, as well as a client selected *IOID* which will be included in the reply.

The IOID MUST be unique on the channel.

Monitor Operation

A Monitor operation is a persistent subscription which is initiated by a *CA_PROTO_EVENT_ADD* request and terminated with a *CA_PROTO_EVENT_CANCEL* request.

Both CA_PROTO_EVENT_ADD and CA_PROTO_EVENT_CANCEL messages include a channel *SID* as well as a client selected *SubscriptionID*.

The SubscriptionID MUST be unique on the channel.

When a subscription is created a server SHOULD immediately send a CA_PROTO_EVENT_ADD reply with the present value of the Channel if such a value is available.

After a CA_PROTO_EVENT_CANCEL request is received, a server MUST send one final CA_PROTO_EVENT_ADD reply with a zero payload size. Before a CA_PROTO_EVENT_CANCEL request is received, a server MUST NOT send a CA_PROTO_EVENT_ADD reply with a zero payload size.

Errors

Any client message MAY result in an *CA_PROTO_ERROR* reply from a server.

Data Count in Gets and Monitors

Prior to `CA_V413`, the element count in a CA_PROTO_EVENT_ADD or CA_PROTO_READ_NOTIFY reply MUST be the same as given in the corresponding CA_PROTO_EVENT_ADD or CA_PROTO_READ_NOTIFY request. A request for zero elements MUST result in an ECA_BADCOUNT exception. If a server can not provide all of the elements requested, then it fills out the message body with null bytes.

Beginning in `CA_V413`, a request for zero elements is valid. The element count in a reply is then the number of elements the server could provide (perhaps zero).

The element count in a reply MUST NOT exceed the maximum element count on the channel.

This dynamic array size feature creates a potential ambiguity in the protocol if the number of bytes in a CA_PROTO_EVENT_ADD reply is zero.

Therefore it is RECOMMENDED that clients not create dynamic monitors for the plain DBR_* types. Clients needing to create such monitors are RECOMMENDED to promote the type to the corresponding DBR_STS_* (the extra meta-data can be ignored for internal processing). Then a zero element count has a non-zero body size.

Note to implementers. RSRV will always give at least one element in CA_PROTO_EVENT_ADD replies. libca will silently ignore CA_PROTO_EVENT_ADD replies with zero size before a CA_PROTO_EVENT_CANCEL request is received.

1.29.6 Data Types

This section defines all primitive data types employed by CA, as well as their C/C++ equivalents. These data types are referred to in the subsequent sections.

Type Name	C/C++	Description
`BYTE`	`char`	Signed 8-bit integer.
`UBYTE`	`unsigned char`	Unsigned 8-bit integer.
`INT16`	`short`	Signed 16-bit integer.
`UINT16`	`unsigned short`	Unsigned 16-bit integer.
`INT32`	`int`	Signed 32-bit integer.
`UINT32`	`unsigned int`	Unsigned 32-bit integer.
`FLOAT`	`float`	IEEE 32-bit float.
`DOUBLE`	`double`	IEEE 64-bit float.
`STRING[`	`char[]`	Array of `UBYTE`s. If `[n]` is specified, it indicates maximum allowed number of characters in this string including (if necessary) termination character.
`TIMESTA`	None	Timestamp represented with two `UINT32` values. First is number of seconds since 0000 Jan 1, 1990. Second is number of nanoseconds within second

All values are transmitted over the network in big-endian (network) order. For example: `UINT32` 3145 (`0x00000C49`) would be sent over the network represented as `00 00 0C 49`.

1.29.7 Messages

Message Structure

All Channel Access messages are composed of a **header**, followed by the **payload**.

Header is always present. The command ID and payload size fields have a fixed meaning. Other header fields carry command-specific meaning. If a field is not used within a certain message, its value MUST be zeroed.

Total size of an individual message is limited. With CA versions older than `CA_V49`, the maximum message size is limited to 16384 (`0x4000`) bytes. Out of these, header has a fixed size of 16 (`0x10`) bytes, with the payload having a maximum size of 16368 (`0x3ff0`) bytes.

Versions `CA_V49` and higher may use the **extended message form**, which allows for larger payloads. The extended message form is indicated by the header fields `Payload Size` and `Data Count` being set to `0xffff` and 0, respectively. Real payload size and data count are then given as `UINT32` type values immediately following the header. Maximum message size is limited by 32-bit unsigned integer representation, 4294967295 (`0xffffffff`). Maximum payload size is limited to 4294967255 (`0xfffffffe7`).

For compatibility, extended message form should only be used if payload size exceeds the pre- `CA_V49` message size limit of 16368 bytes.

Header

Table 1. Standard Message Header

0	1	2	3	4	5	6	7
Command		Payload		Data Type		Data Count	
Parameter 1				Parameter 2			

Table 2. Extended Message Header

0	1	2	3	4	5	6	7
Command		0xFFFF		Data Type		0x0000	
Parameter 1				Parameter 2			
Payload size				Data count			

Names of header fields are based on their most common use. Certain messages will use individual fields for purposes other than those described here. These variations are documented for each message individually. All of values in header are unsigned integers.

Generic header fields:

Parameter	Type		Description
Command	`UINT16`		Identifier of the command this message requests. The meaning of other header fields and the payload depends on the command.
Payload Size	`UINT16` or `UINT32`		Size of the payload (in bytes). MUST not exceed `0x4000` for UDP.
Data Type	`UINT16`		Identifier of the data type carried in the payload. Data types are defined in section *Payload Data Types*.
Data Count	`UINT16` or `UINT32`		Number of elements in the payload.
Parameter 1	`UINT32`		Command dependent parameter.
Parameter 2	`UINT32`		Command dependent parameter.

Payload

The structure of the payload depends on the type of the message. The size of the payload matches the `Payload Size` header field.

Message payloads MUST be padded to a length which is a multiple of 8 bytes. Zero padding is RECOMMENDED.

Message Identifiers

Some fields in messages serve as identifiers. These fields serve as identification tokens in within the context of a circuit (TCP connection). The RECOMMENDED scheme for allocating these values is to create them sequentially starting at 0. All IDs are represented with `UINT32`.

Overflow of all identifiers MUST be handled! A long running applications might use more than 2**32 of some identifier type (typically IOID).

CID - Client ID

A CID is the client selected identifier for a channel. A CID MUST be unique for a single Circuit.

Clients MUST not send a request with a CID which is not associated with an *active Channel*.

Servers MUST ignore any request which does not include the CID of an active channel without closing the Circuit.

A CID is found in the Parameter 1 field of *CA_PROTO_ERROR*, *CA_PROTO_CREATE_CHAN*, *CA_PROTO_ACCESS_RIGHTS*, *CA_PROTO_CREATE_CH_FAIL*, and *CA_PROTO_SERVER_DISCONN* messages. And in the Parameter 2 field of *CA_PROTO_CLEAR_CHANNEL* message.

SID - Server ID

A SID is the server selected identifier for a channel. A SID MUST be unique for a single Circuit.

Servers MUST not send a request with a SID which is not associated with an *active Channel*.

Clients MUST ignore any request which does not include the SID of an active channel without closing the Circuit.

A SID is found in the Parameter 1 field of *CA_PROTO_EVENT_ADD*, *CA_PROTO_EVENT_CANCEL*, *CA_PROTO_READ_NOTIFY*, *CA_PROTO_WRITE_NOTIFY*, *CA_PROTO_WRITE*, *CA_PROTO_CLEAR_CHANNEL*, and *CA_PROTO_CREATE_CHAN* (reply only) messages,

Subscription ID

A SubscriptionID is the client selected identifier for a subscription. A CID MUST be unique for a single Circuit.

A SubscriptionID is found in the Parameter 2 field of *CA_PROTO_EVENT_ADD* and *CA_PROTO_EVENT_CANCEL* messages.

IOID

An IOID is the client selected identifier for a Get or Put operation. An IOID MUST be unique for a single message type on a single Circuit.

It is possible though NOT RECOMMENDED to use the same IOID concurrently in a CA_PROTO_WRITE, a CA_PROTO_READ_NOTIFY, and a CA_PROTO_WRITE_NOTIFY request.

An IOID is found in the Parameter 2 field of *CA_PROTO_READ_NOTIFY*, *CA_PROTO_WRITE_NOTIFY*, and *CA_PROTO_WRITE* messages.

Search ID

A SearchID is a client selected identifier for a PV name search. A SearchID must be unique for each client endpoint sending requests.

Due to the nature of UDP it is possible for datagrams to be duplicated. Several CA_PROTO_SEARCH messages with the same SearchID MAY be considered to be duplicates, and only one used.

1.29.8 Commands (TCP and UDP)

The following commands are sent as either UDP datagrams or TCP messages. Some of the messages are also used within the context of a Virtual Circuit (TCP connection).

CA_PROTO_VERSION

Command	CA_PROTO_VERSION
ID	0 (0x00)
Description	Exchanges client and server protocol versions and desired circuit priority. MUST be the first message sent, by both client and server, when a new TCP (Virtual Circuit) connection is established. It is also sent as the first message in UDP search messages.

Request

Field	Value	Description
Command	0	Command identifier for CA_PROTO_VERSION.
Payload size	0	Must be 0.
Priority	Desired priority	Virtual circuit priority.
Version	Version number	Minor protocol version number. Only used when sent over TCP.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 3. Header

Version	Comment
>= CA_V41	Server will send response immediately after establishing a virtual circuit.
< CA_V41	Message does not include minor version number (it is always 0) and is interpreted as an echo command that carries no data. Version exchange is performed immediately after *CA_PROTO_CREATE_CHAN*.

Table: Table 4. Compatibility

Comments

- Priority indicates the server's dispatch scheduling priority which might be implemented by a circuit dedicated thread's scheduling priority in a preemptive scheduled OS.
- Due to a buffering bug, RSRV implementing < CA_V411 did not send CA_PROTO_VERSION immediately on connection, but rather when some other response triggers a buffer flush.

Response

Field	Value	Description
Command	0	Command identifier for CA_PROTO_VERSION.
Reserved	0	Must be 0.
Priority	0	Must be 0.
Version	Version number	Minor protocol version number. Only used when sent over TCP.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 5. Header

Version	Comment
>= CA_V411	Server will not respond to request, but send response immediately after establishing a virtual circuit.
< CA_V411	Message does not include minor version number (it is always 0).

Table: Table 6. Compatibility

CA_PROTO_SEARCH

Command	CA_PROTO_SEARCH
ID	6 (0x06)
Description	Searches for a given channel name. Sent over UDP or TCP.

Request

Field	Value	Description
Command	6	Command identifier for CA_PROTO_SEARCH.
Payload Size	>= 0	Padded size of channel name.
Reply	Reply Flag	*Search Reply Flag*, indicating whether failed search response should be returned.
Version	Version Number	Client minor protocol version number.
SearchID		Client allocated Search identifier.
SearchID		Client allocated Search identifier.

Table: Table 7. Header

Name	Type	Value	Description
Channel name	STRING		Name of channel to search for.

Table: Table 8. Payload

Comments

- Sent as a UDP datagram.
- It is illegal to specify DO_REPLY flag whenever the message is sending as UDP datagram, regardless of whether broadcast or multicast is used.
- SearchID will be allocated by the client before this message is sent.
- SearchID field value is duplicated.
- Reply flag will be generally DONT_REPLY when searching using broadcast and DO_REPLY when searching using unicast. When DO_REPLY is set, server will send a *CA_PROTO_NOT_FOUND* message indicating it does not have the requested channel.

Response

Field	Value	Description
Command	6	Command identifier for `CA_PROTO_SEARCH`.
Payload Size	8	Payload size is constant.
Data Type	Port number	TCP Port number of server that responded.
Data Count	0	Must be 0.
SID or IP	`0xffffffff`	Temporary *SID* (deprecated) or server IP address.
SearchID		Client allocated Search identifier.

Table: Table 9. Header

Name	Type	Value	Description
Server protocol version	`UINT16`		Server protocol version.

Table: Table 10. Payload

Comments

- Received as UDP datagram.
- Search ID field value (CID) is copied from the request.
- Before `CA_V411` the SID/IP field will always have the value of `0xffffffff` and the server IP address is assumed to be the senders IP.
- Starting with `CA_V411` the server's IP address is encoded in the SID/IP field if it differs from the sender's IP, or `0xffffffff` if it is the same.
- The port number included in the header is the **TCP** port of the server. Two servers on the same host can share a UDP port number, but not a TCP port number. Therefore, the port the client needs to connect to in that situation may not be the same as expected if this field in the response is not used.

CA_PROTO_NOT_FOUND

Command	`CA_PROTO_NOT_FOUND`
ID	14 (`0x0E`)
Description	Indicates that a channel with requested name does not exist. Sent in response to *CA_PROTO_SEARCH*, but only when its `DO_REPLY` flag was set. Sent over UDP.

Response

Field	Value	Description
Command	14	Command identifier for CA_PROTO_NOT_FOUND.
Reserved	0	Must be 0.
Reply Flag	DO_REPLY	Same reply flag as in request: always DO_REPLY.
Version	Same as request	Client minor protocol version number.
SearchID		Client allocated Search identifier.
SearchID		Client allocated Search identifier.

Table: Table 11. Header

Comments

- Contents of the header are identical to the request.
- SearchID fields are duplicated.
- Original request payload is not returned with the response.

CA_PROTO_ECHO

Command	CA_PROTO_ECHO
ID	23 (0x17)
Description	Connection verify used by CA_V43. Sent over TCP.

Request

Field	Value	Description
Command	23	Command identifier for CA_PROTO_ECHO.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 12. Header

Response

Field	Value	Description
Command	23	Command identifier for CA_PROTO_ECHO.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 13. Header

1.29.9 Commands (UDP)

The following commands are sent as UDP datagrams.

CA_PROTO_RSRV_IS_UP

Command	CA_PROTO_RSRV_IS_UP
ID	13 (0x0D)
Description	Beacon sent by a server when it becomes available. Beacons are also sent out periodically to announce the server is still alive. Another function of beacons is to allow detection of changes in network topology. Sent over UDP.

Response

Field	Value	Description
Command	13	Command identifier for CA_PROTO_RSRV_IS_UP.
Reserved	0	Must be 0.
Version	Version number	CA protocol version
Server port	>= 0	TCP Port the server is listening on.
BeaconID	Sequential integers	Sequential Beacon ID.
Address	0 or IP	May contain IP address of the server.

Table: Table 14. Header

Comments

- IP field may contain IP of the server. If IP is not present (field Address value is 0), then IP may be substituted by the receiver of the packet (usually repeater) if it is capable of identifying where this packet came from. Any non-zero address must be interpreted as server's IP address.
- BeaconIDs are useful in detecting network topology changes. In certain cases, same packet may be routed using two different routes, causing problems with datagrams. If multiple beacons are received from the same server with same BeaconID, multiple routes are the cause.
- If a server is restarted, it will most likely start sending BeaconID values from beginning (0). Such situation must be anticipated.

CA_REPEATER_CONFIRM

Command	CA_REPEATER_CONFIRM
ID	17 (0x11)
Description	Confirms successful client registration with repeater. Sent over UDP.

Response

Field	Value	Description
Command	17	Command identifier for CA_REPEATER_CONFIRM.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Repeater address	IP address	Address with which the registration succeeded.

Table: Table 15. Header

Comments

- Since repeater can bind to different local address, its IP is reported in Repeater address. This address will be either 0.0.0.0 or 127.0.0.1.

CA_REPEATER_REGISTER

Command	CA_REPEATER_REGISTER
ID	24 (0x18)
Description	Requests registration with the repeater. Repeater will confirm successful registration using CA_REPEATER_CONFIRM. Sent over TCP.

Request

Field	Value	Description
Command	CA_REPEATER_REGISTER	Command identifier
Reserved	0	Must be 0
Reserved	0	Must be 0
Reserved	0	Must be 0
Reserved	0	Must be 0
Client IP address	IP address	IP address on which the client is listening

Table: Table 16. Header

1.29.10 Commands (TCP)

The following commands are used within the context of Virtual Circuit and are sent using TCP.

CA_PROTO_EVENT_ADD

Command	CA_PROTO_EVENT_ADD
ID	1 (0x01)
Description	Creates a subscription on a channel, allowing the client to be notified of changes in value. A request will produce at least one response. Sent over TCP.

Request

Field	Value	Description
Command	1	Command identifier for CA_PROTO_EVENT_ADD
Payload Size	16	Payload size is constant
Data Type		Desired DBR type of the return value.
Data Count	>= 0	Desired number of elements
SID	SID of the channel.	SID of the channel on which to register this subscription. See *SID - Server ID*.
SubscriptionID	Client provided Subscription ID	Subscription ID identifying this subscription.See *Subscription ID*.

Table: Table 17. Header

Payload

Name	Type	Value	Description
Low val	FLOAT32	0.0	Low value
High val	FLOAT32	0.0	High value
To val	FLOAT32	0.0	To value
Mask	UINT16	Monitor mask	*Mask* indicating which events to report

Comments

- All payload fields except Mask are initialized to 0 and are present only for backward compatibility.
- Successful subscription will result in an immediate response with the current value. Additional responses will be sent as the change occurs based on the Mask parameter.
- Mask defines a filter on which events will be sent.
- A subscription should be destroyed when no longer needed to reduce load on server. See *CA_PROTO_EVENT_CANCEL*.

Response

Field	Value	Description
Command	1	Command identifier for CA_PROTO_EVENT_ADD
Payload Size	>= 0	Size of the response.
Data Type	same as request	Payload data type.
Data Count	same as request	Payload data count.
Status code	One of ECA codes	*Status code* (ECA_NORMAL on success).
SubscriptionID	same as request	Subscription ID

Table: Table 18. Header

Name	Type	Value	Description
Values	DBR		Value stored as DBR type specified in Data Type field. See *Payload Data Types*.

Table: Table 19. Payload

Comments

- Response data type and count match that of the request.
- To confirm successful subscription, first response will be sent immediately. Additional responses will be sent as the change occurs based on mask parameters.

CA_PROTO_EVENT_CANCEL

Command	CA_PROTO_EVENT_CANCEL
ID	2 (0x02)
Description	Clears event subscription. This message will stop event updates for specified channel. Sent over TCP.

Request

Field	Value	Description
Command	2	Command identifier for CA_PROTO_EVENT_CANCEL.
Payload Size	0	Must be 0.
Data Type		Same value as in corresponding *CA_PROTO_EVENT_ADD*.
Data Count	>= 0	Same value as in corresponding *CA_PROTO_EVENT_ADD*.
SID	SID of channel	Same value as in corresponding *CA_PROTO_EVENT_ADD*.
SubscriptionID	Subscription ID	Same value as in corresponding *CA_PROTO_EVENT_ADD*.

Table: Table 20. Header

Comments

- Both SID and SubscriptionID are used to identify which subscription on which monitor to destroy.
- Actual data type and count values are not important, but should be the same as used with corresponding *CA_PROTO_EVENT_ADD*.

Response

Field	Value	Description
Command	1	Command identifier for `CA_PROTO_EVENT_ADD`.
Payload Size	0	Must be 0.
Data Type	Same as request.	Same value as `CA_PROTO_EVENT_ADD` request.
Data Count	0	Must be 0.
SID	Same as request.	Same value as `CA_PROTO_EVENT_ADD` request.
SubscriptionID	Same as request.	Same value as `CA_PROTO_EVENT_ADD` request.

Table: Table 21. Header

Comments

- Notice that the response has *CA_PROTO_EVENT_ADD* command identifier!
- Regardless of data type and count, this response has no payload.

CA_PROTO_READ

Command	`CA_PROTO_READ`
ID	3 (`0x03`)
Description	Read value of a channel. Sent over TCP.

Deprecated since protocol version 3.13.

Request

Field	Value	Description
Command	3	Command identifier for `CA_PROTO_READ_NOTIFY`.
Payload Size	0	Must be 0.
Data Type	DBR type	Desired type of the return value.
Data Count	>= 0	Desired number of elements to read.
SID	Channel SID	SID of the channel to read.
IOID	Client provided IOID	IOID of this operation.

Table: Table 22. Header

Comments

- Channel from which to read is identified using SID.
- Response will contain the same IOID as the request, making it possible to distinguish multiple responses.

Response

Field	Value	Description
Command	3	Command identifier for `CA_PROTO_READ_NOTIFY`.
Payload size	Size of payload	Size of DBR formatted data in payload.
Data type	DBR type	Payload format.
Data count	>= 0	Payload element count.
SID	Same as request	SID of the channel.
IOID	Same as request	IOID of this operation.

Table: Table 23. Header

Name	Type	Value	Description
DBR formatted data	DBR	DBR formatted data	Value stored as DBR type specified in Data type field. Data count specifies number of elements of DBR value field.

Table: Table 24. Payload

`CA_PROTO_WRITE`

Command	`CA_PROTO_WRITE`
ID	4 (`0x04`)
Description	Writes new channel value. Sent over TCP.

Request

Field	Value	Description
Command	`CA_PROTO_WRITE`	Command identifier
Payload size	Size of DBR formatted payload	Size of padded payload
Data type	DBR type	Format of payload
Data count	`ELEMENT_COUNT`	Number of elements in payload
SID	SID provided by server	Server channel ID
IOID	Client provided IOID	Request ID

Table: Table 25. Header

Name	Type	Value	Description
DBR formatted data	DBR	DBR formatted data	Value stored as DBR type specified in Data type field. Data count specifies number of elements of DBR value field.

Table: Table 26. Payload

Comments

- There is no response to this command.

CA_PROTO_SNAPSHOT

Command	CA_PROTO_SNAPSHOT
ID	5 (0x05)
Description	Obsolete.

CA_PROTO_BUILD

Command	CA_PROTO_BUILD
ID	7 (0x07)
Description	Obsolete.

CA_PROTO_EVENTS_OFF

Command	CA_PROTO_EVENTS_OFF
ID	8 (0x08)
Description	Disables a server from sending any subscription updates over this virtual circuit. Sent over TCP. This mechanism is used by clients with slow CPU to prevent congestion when they are unable to handle all updates received. Effective automated handling of flow control is beyond the scope of this document.

Request

Field	Value	Description
Command	8	Command identifier for CA_PROTO_EVENTS_OFF
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 27. Header

Comments

- This request will disable sending of subscription updates on the server to which it is sent.
- Command applies to a single virtual circuit, so having multiple priority virtual circuit connections to the server would only affect the one on which the message is sent.
- No response will be sent for this request.

CA_PROTO_EVENTS_ON

Command	CA_PROTO_EVENTS_ON
ID	9 (0x09)
Description	Enables the server to resume sending subscription updates for this virtual circuit. Sent over TCP. This mechanism is used by clients with slow CPU to prevent congestion when they are unable to handle all updates received. Effective automated handling of flow control is beyond the scope of this document.

Request

Field	Value	Description
Command	9	Command identifier for CA_PROTO_EVENTS_ON
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 28. Header

Comments

- This request will enable sending of subscription updates on the server to which it is sent.
- Command applies to a single virtual circuit, so having multiple priority virtual circuit connections to the server would only affect the one on which the message is sent.
- No response will be sent for this request.

CA_PROTO_READ_SYNC

Command	CA_PROTO_READ_SYNC
ID	10 (0x0A)
Description	**Deprecated since protocol version 3.13.**

Request

Field	Value	Description
Command	10	Command identifier for CA_PROTO_READ_SYNC.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 29. Header

CA_PROTO_ERROR

Command	CA_PROTO_ERROR
ID	11 (0x0B)
Description	Sends error message and code. This message is only sent from server to client in response to any request that fails and does not include error code in response. This applies to all asynchronous commands. Error message will contain a copy of original request and textual description of the error. Sent over UDP.

Response

Field	Value	Description
Command	11	Command identifier for CA_PROTO_ERROR
Payload Size		Size of the request header that triggered the error plus size of the error message.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
CID	Channel CID	*CID* of the channel for which request failed.
Status Code	One of ECA codes	*Error status code*.

Table: Table 30. Header

Name	Type	Value	Description
Original Request	Message Header		Header of the request that caused the error.
Error Message	STRING		A null-terminated string conveying the error message.

Table: Table 31. Payload

Comments

- Complete exception report is returned. This includes error message code, CID of channel on which the request failed, original request and string description of the message.
- CID value depends on original request and may not actually identify a channel.
- First part of payload is original request header with the same structure as sent. Any payload that was part of this request is not included. Textual error message starts immediately after the header.

CA_PROTO_CLEAR_CHANNEL

Command	CA_PROTO_CLEAR_CHANNEL
ID	12 (0x0C)
Description	Clears a channel. This command will cause server to release the associated channel resources and no longer accept any requests for this SID/CID.

Request

Field	Value	Description
Command	12	Command identifier of CA_PROTO_CLEAR_COMMAND
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
SID	SID of the channel	SID of channel to clear.
CID	CID of the channel	CID of channel to clear.

Table: Table 32. Header

Response

Field	Value	Description
Command	12	Command identifier of CA_PROTO_CLEAR_COMMAND
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
SID	Same as request	SID of cleared channel.
CID	Same as request	CID of cleared channel.

Table: Table 33. Header

Comments

- Server responds immediately and only then releases channel resources.
- Once a channel with a given SID has been cleared, any request sent with this SID will fail.
- Sent over TCP.

CA_PROTO_READ_NOTIFY

Command	CA_PROTO_READ_NOTIFY
ID	15 (0x0F)
Description	Read value of a channel. Sent over TCP.

Request

Field	Value	Description
Command	15	Command identifier for CA_PROTO_READ_NOTIFY.
Payload Size	0	Must be 0.
Data Type	DBR type	Desired type of the return value.
Data Count	>= 0	Desired number of elements to read.
SID	Channel SID	SID of the channel to read.
IOID	Client provided IOID	IOID of this operation.

Table: Table 34. Header

Comments

- Channel from which to read is identified using SID.
- Response will contain the same IOID as the request, making it possible to distinguish multiple responses.

Response

Field	Value	Description
Command	15	Command identifier for `CA_PROTO_READ_NOTIFY`.
Payload size	Size of payload	Size of DBR formatted data in payload.
Data type	DBR type	Payload format.
Data count	>= 0	Payload element count.
SID	Same as request	SID of the channel.
IOID	Same as request	IOID of this operation.

Table: Table 35. Header

Name	Type	Value	Description
DBR formatted data	DBR	DBR formatted data	Value stored as DBR type specified in Data type field. Data count specifies number of elements of DBR value field.

Table: Table 36. Payload

`CA_PROTO_READ_BUILD`

Command	`CA_PROTO_READ_BUILD`
ID	16 (`0x10`)
Description	Obsolete

Request

`CA_PROTO_CREATE_CHAN`

Command	`CA_PROTO_CREATE_CHAN`
ID	18 (`0x12`)
Description	Requests creation of channel. Server will allocate required resources and return initialized SID. Sent over TCP.

Request

Field	Value	Description
Command	18	Command identifier for `CA_PROTO_CREATE_CHAN`
Payload size	Size of payload	Padded length of channel name.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
CID	Channel CID	CID of the channel to create.
Client version	Version number	Client minor protocol version.

Table: Table 37. Header

Payload

|[options="header"]

Name	Type	Value	Description
Channel name	`STRING`		Name of channel to create.

Comments

- CID sent should be the same as used with `CA_PROTO_SEARCH`.

Response

Field	Value	Description
Command	`CA_PROTO_CREATE_CHAN`	
Payload size	0	Must be 0
Data type	DBR type	Native channel data type
Data count	>= 0	Native channel data count
CID	Same as request	Channel client ID
SID	SID provided by server	Channel server ID

Table: Table 38. Header

Comments

- SID will be associated with CID on the server and will be reused sending certain commands that require it as a parameter.
- SID will be valid until the channel is cleared using `CA_PROTO_CLEAR` or server destroys the PV the channel references.

CA_PROTO_WRITE_NOTIFY

Command	CA_PROTO_WRITE_NOTIFY
ID	19 (0x13)
Description	Writes new channel value. Sent over TCP.

Request

Field	Value	Description
Command	CA_PROTO_WRITE_NOTIFY	Command identifier
Payload size	Size of DBR formatted payload	Size of padded payload
Data type	DBR type	Format of payload
Data count	ELEMENT_COUNT	Number of elements in payload
SID	SID provided by server	Server channel ID
IOID	Client provided IOID	Request ID

Table: Table 39. Header

Name	Type	Value	Description
DBR formatted data	DBR	DBR formatted data	Value stored as DBR type specified in Data type field. Data count specifies number of elements of DBR value field.

Table: Table 40. Payload

Response

Field	Value	Description
Command	CA_PROTO_WRITE_NOTIFY	Command identifier
Payload size	0	Must be 0
Data type	Same as request	Format of data written
Data count	Same as request	Number of elements written
Status	Status code	Status of write success
IOID	Same as request	Request ID

Table: Table 41. Header

CA_PROTO_CLIENT_NAME

Command	CA_PROTO_CLIENT_NAME
ID	20 (0x14)
Description	Sends local username to virtual circuit peer. This name identifies the user and affects access rights.

Request

Field	Value	Description
Command	CA_PROTO_CLIENT_NAME	Command identifier
Payload size	>=0	Length of string in payload
Reserved	0	Must be 0
Reserved	0	Must be 0
Reserved	0	Must be 0
Reserved	0	Must be 0

Table: Table 42. Header

Name	Type	Value	Description
User name	STRING		0-terminated username string

Table: Table 43. Payload

Comments

- This is a one-way message and will not receive response.
- String in payload must be 0 padded to a length that is multiple of 8.
- Sent over TCP.

CA_PROTO_HOST_NAME

Command	CA_PROTO_HOST_NAME
ID	21 (0x15)
Description	Sends local host name to virtual circuit peer. This name will affect access rights. Sent over TCP.

Request

Field	Value	Description
Command	21	Command identifier for CA_PROTO_HOST_NAME.
Payload size	Size of payload	Length of host name string.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.

Table: Table 44. Header

Name	Type	Value	Description
Host name	STRING		Client host name.

Table: Table 45. Payload

Comments

- This is one-way message and will receive no response.

CA_PROTO_ACCESS_RIGHTS

Command	CA_PROTO_ACCESS_RIGHTS
ID	22 (0x16)
Description	Notifies of access rights for a channel. This value is determined based on host and client name and may change during runtime. Client cannot change access rights nor can it explicitly query its value, so last received value must be stored.

Response

Field	Value	Description
Command	22	Command identifier for CA_PROTO_ACCESS_RIGHTS.
Payload size	0	Must be 0.
Reserved	0	Must be 0.
Reserved	0	Must be 0.
CID	Channel CID	Channel affected by change.
Access Rights	Access Rights	*Access rights* for given channel.

Table: Table 46. Header

Comments

- Access Rights affect CA_PROTO_READ_NOTIFY, CA_PROTO_WRITE_NOTIFY and CA_PROTO_WRITE.
- CA_PROTO_ACCESS_RIGHTS will be sent immediately after a channel is created using CA_PROTO_CREATE_CHAN. If they change during runtime, this message sent to report new value.
- Changes are only sent to currently connected channels, since it requires valid CID.
- Sent over TCP.

CA_PROTO_SIGNAL

Command	CA_PROTO_SIGNAL
ID	25 (0x19)
Description	Obsolete.

CA_PROTO_CREATE_CH_FAIL

Command	CA_PROTO_CREATE_CH_FAIL
ID	26 (0x1A)
Description	Reports that channel creation failed. This response is sent to when channel creation in CA_PROTO_CREATE_CHAN fails.

Response

Field	Value	Description
Command	CA_PROTO_CREATE_CH_FAIL	Command identifier
Reserved	0	Must be 0
Reserved	0	Must be 0
Reserved	0	Must be 0
CID	Same as request	Client channel ID
Reserved	0	Must be 0

Table: Table 47. Header

Comments

- Sent over TCP.

CA_PROTO_SERVER_DISCONN

Command	CA_PROTO_SERVER_DISCONN
ID	27 (0x1B)
Description	Notifies the client that server has disconnected the channel. This may be since the channel has been destroyed on server. Sent over TCP.

Response

Field	Value	Description
Command	CA_PROTO_SERVER_DISCONN	Command identifier
Reserved	0	Must be 0
Reserved	0	Must be 0
Reserved	0	Must be 0
CID	CID provided by client	CID that was provided during CA_PROTO_CREATE_CHAN
Reserved	0	Must be 0

Table: Table 48. Header

1.29.11 Payload Data Types

Channel access defines special structures to transferring data. These types are organized in typed hierarchies with loose inheritance. There are six basic data types: DBR_STRING, DBR_SHORT, DBR_FLOAT, DBR_ENUM, DBR_CHAR, DBR_LONG and DBR_DOUBLE. The type DBR_INT is present as an alias for DBR_SHORT. Each of these types can represent an array of elements.

In addition to element values, some DBR types include meta-data. These types are status (DBR_STS_*), time stamp (DBR_TIME_*), graphic (DBR_GR_*) and control (DBR_CTRL_*). All these structures contain value as the last field.

All DBR data MUST be zero padded to ensure that message body length is a multiple of 8 bytes. Therefore, when receiving a message, it is necessary to use the DBR type and element count to determine the number of body bytes to use. Additional body bytes MUST be ignored.

In addition to zero padding at the end of the message, some padding is placed between the meta-data and the value array.

The following table lists the identifier, meta-data size, padding between meta-data and value, and value element sizes of each DBR type.

Name	ID	Meta size	padding	Element size
DBR_STRING	0	0	0	40
DBR_INT	1	0	0	2
DBR_SHORT	1	0	0	2
DBR_FLOAT	2	0	0	4
DBR_ENUM	3	0	0	2
DBR_CHAR	4	0	0	1
DBR_LONG	5	0	0	4
DBR_DOUBLE	6	0	0	8
DBR_STS_STRING	7	4	0	40
DBR_STS_INT	8	4	0	2
DBR_STS_SHORT	8	4	0	2
DBR_STS_FLOAT	9	4	0	4
DBR_STS_ENUM	10	4	0	2
DBR_STS_CHAR	11	4	1	1
DBR_STS_LONG	12	4	0	4
DBR_STS_DOUBLE	13	4	4	8
DBR_TIME_STRING	14	12	0	40

continues on next page

Table 3 – continued from previous page

Name	ID	Meta size	padding	Element size
DBR_TIME_INT	15	12	2	2
DBR_TIME_SHORT	15	12	2	2
DBR_TIME_FLOAT	16	12	0	4
DBR_TIME_ENUM	17	12	2	2
DBR_TIME_CHAR	18	12	3	1
DBR_TIME_LONG	19	12	0	4
DBR_TIME_DOUBLE	20	12	4	8
DBR_GR_STRING	21	4	0	40
DBR_GR_INT	22	GR_INT	0	2
DBR_GR_SHORT	22	GR_INT	0	2
DBR_GR_FLOAT	23	GR_REAL	2	4
DBR_GR_ENUM	24	GR_ENUM	0	2
DBR_GR_CHAR	25	GR_INT	1	1
DBR_GR_LONG	26	GR_INT	0	4
DBR_GR_DOUBLE	27	GR_REAL	0	8
DBR_CTRL_STRING	28	4	0	40
DBR_CTRL_INT	29	CTRL_INT	0	2
DBR_CTRL_SHORT	29	CTRL_INT	0	2
DBR_CTRL_FLOAT	30	CTRL_REAL	0	2
DBR_CTRL_ENUM	31	GR_ENUM	0	2
DBR_CTRL_CHAR	32	CTRL_INT	1	1
DBR_CTRL_LONG	33	CTRL_INT	0	4
DBR_CTRL_DOUBLE	34	CTRL_REAL	0	8
DBR_PUT_ACKT	35	?	?	2
DBR_PUT_ACKS	36	?	?	2
DBR_STSACK_STRING	37	?	?	40
DBR_CLASS_NAME	38	?	?	40

Table: Table 49. DBRs

DBR_STS_* meta-data

Alarm meta-data. Length: 4 bytes

```
struct metaSTS {
    epicsInt16 status;
    epicsInt16 severity;
};
```

DBR_TIME_* meta-data

Alarm and time stamp meta-data. Length: 12 bytes

```
struct metaTIME {
    epicsInt16 status;
    epicsInt16 severity;
    epicsInt32 secondsSinceEpoch;
    epicsUInt32 nanoSeconds;
};
```

Note that the EPICS Epoch is 1990-01-01T00:00:00Z. This is 631152000 seconds after the POSIX Epoch of 1970-01-01T00:00:00Z.

DBR_GR_SHORT meta-data

Alarm and integer display meta-data (no timestamp). Length: ?? bytes

```
struct metaGR_INT {
    epicsInt16 status;
    epicsInt16 severity;
    char units[8];
    epicsInt16 upper_display_limit;
    epicsInt16 lower_display_limit;
    epicsInt16 upper_alarm_limit;
    epicsInt16 upper_warning_limit;
    epicsInt16 lower_warning_limit;
    epicsInt16 lower_alarm_limit;
};
```

DBR_GR_CHAR meta-data

Alarm and integer display meta-data (no timestamp). Length: ?? bytes

```
struct metaGR_INT {
    epicsInt16 status;
    epicsInt16 severity;
    char units[8];
    epicsInt8 upper_display_limit;
    epicsInt8 lower_display_limit;
    epicsInt8 upper_alarm_limit;
    epicsInt8 upper_warning_limit;
    epicsInt8 lower_warning_limit;
    epicsInt8 lower_alarm_limit;
};
```

DBR_GR_FLOAT meta-data

Alarm and floating point display meta-data (no timestamp). Length: ?? bytes

```
struct metaGR_FLOAT {
    epicsInt16 status;
    epicsInt16 severity;
    epicsInt16 precision;
    epicsInt16 padding;
    char units[8];
    epicsFloat32 upper_display_limit;
    epicsFloat32 lower_display_limit;
    epicsFloat32 upper_alarm_limit;
    epicsFloat32 upper_warning_limit;
    epicsFloat32 lower_warning_limit;
```

(continues on next page)

(continued from previous page)

```
    epicsFloat32 lower_alarm_limit;
};
```

DBR_GR_DOUBLE meta-data

Alarm and floating point display meta-data (no timestamp). Length: ?? bytes

```
struct metaGR_FLOAT {
    epicsInt16 status;
    epicsInt16 severity;
    epicsInt16 precision;
    epicsInt16 padding;
    char units[8];
    epicsFloat64 upper_display_limit;
    epicsFloat64 lower_display_limit;
    epicsFloat64 upper_alarm_limit;
    epicsFloat64 upper_warning_limit;
    epicsFloat64 lower_warning_limit;
    epicsFloat64 lower_alarm_limit;
};
```

GR_ENUM and CTRL_ENUM meta-data

Alarm and enumerated display meta-data (no timestamp). Length: ?? bytes

```
struct metaGR_ENUM {
    epicsInt16 status;
    epicsInt16 severity;
    epicsInt16 number_of_string_used;
    char strings[16][26];
};
```

The `strings` field is an array of 16 string of 26 characters. The `number_of_string_used` gives the number of entries in the `strings` field which are valid. Additional `strings` should be ignored, even if they contain non-null bytes.

1.29.12 Constants

Port numbers

Although there is no requirement as to which port numbers are used by either servers or clients, there are some standard values which must be used as defaults, unless overridden by application.

Port numbers are dependent on protocol versions and are calculated using the following definitions:

`CA_PORT_BASE` = 5056

`CA_SERVER_PORT` = `CA_PORT_BASE` + `MAJOR_PROTOCOL_VERSION` * 2

`CA_REPEATER_PORT` = `CA_PORT_BASE` + `MAJOR_PROTOCOL_VERSION` * 2 + 1

Based on protocol version described in this document (4.11), port numbers used are `CA_SERVER_PORT` = 5064 and `CA_REPEATER_PORT` = 5065.

Since registration of port numbers with IANA and in the interest of compatibility, the version numbers are unlikely to change. Therefore, the port numbers described here (5064 and 5065) may be considered final.

Representation of constants

This section lists various constants, their types and values used by protocol.

Some constants can be combined using logical OR operation. Example: Monitor mask of `DBE_VALUE` and `DBE_ALARM` are combined using `(DBE_VALUE or DBE_ALARM)` resulting in `(1 or 4 == 5)`.

To query whether a certain value is present in such combined value, and operation is used. Example: to query whether `DBE_ALARM` of monitor mask is set, `(DBE_VALUE and MASK > 0)` will return 0 if `DBE_VALUE` is not present, otherwise `DBE_ALARM` is present.

Monitor Mask

Indicates which changes to the value should be reported back to client library. Different values can be combined using logical OR operation.

Type: not defined, depends on the field it is in (usually `UINT16`)

- `DBE_VALUE` - value 1 (`0x01`) - Value change events are reported. Value changes take into consideration a dead band within which the value changes are not reported.
- `DBE_LOG` - value 2 (`0x02`) - Log events are reported. Similar to `DBR_VALUE`, `DBE_LOG` defines a different dead band value that determines frequency of updates.
- `DBE_ALARM` - value 4 (`0x04`) - Alarm events are reported whenever alarm value of the channel changes.
- `DBE_PROPERTY` - value 8 (`0x08`) - Property events are reported when some metadata value associated with the channel changes. (Introduced in EPICS Base 3.14.11).

Notes

- CA Servers SHOULD ignore unknown monitor mask bits.
- Older PCAS versions will respond to unknown bits with `ECA_BADMASK`.

Search Reply Flag

Indicates whether server should reply to failed search messages. If a server does not know about channel name, it has the option of replying to request or ignoring it. Usually, servers contacted through address list will receive request for reply.

Type: not defined, depends on the field it is in (usually `UINT16`).

- `DO_REPLY` - value 10 (`0x0a`) - Server should reply to failed search requests.
- `DONT_REPLY` - value 5 (`0x05`) - Server should ignore failed requests.

Access Rights

Defines access rights for a given channel. Access rights are defined as logical OR'ed values of allowed access.

Type: not defined, depends on the field it is in (usually `UINT16`).

- `CA_PROTO_ACCESS_RIGHT_READ` - value 1 (`0x01`) - Read access is allowed
- `CA_PROTO_ACCESS_RIGHT_WRITE` - value 2 (`0x02`) - Write access is allowed.

As a reference, the following values are valid.

- 0 - No access
- 1 - Read access only
- 2 - Write access only
- 3 - Read and write access

Servers MUST set undefined bits to zero. Clients MUST ignore undefined bits in this field.

1.29.13 Example message

This example shows construction of messages. For details of individual structures, see message and data type reference (`CA_PROTO_READ_NOTIFY` and `DBR_GR_INT16`).

A client will send `CA_PROTO_READ_NOTIFY` message with the following contents.

- Data type: `DBR_GR_INT16`
- Element count: 5
- Server ID: 22 (obtained during channel creation)
- Sequence ID: 56 (each read or write request increases value by one)

The message would be represented as follows:

```
00 0F (command) 00 00 (payload size) 00 16 (data type) 00 05 (element count)
00 00 00 16 (server ID) 00 00 00 38 (sequence ID)
```

Server would respond with success and return requested value with individual `DBR_GR_INT16` fields having the following values.

- Status: `ECA_NORMAL`
- Severity: `NO_ALARM` (0)

```
00 0f (command) 00 20 (payload size) 00 16 (data type) 00 05 (element count)
00 00 00 16 (server ID) 00 00 00 38 (sequence ID)
00 05  00 02  43 6f 75 6e 74 73 00 00  00 0a  00 00
00 08  00 06  00 04  00 02  00 00  00 00  00 00 00 00
    8      6      4      2      0      0      0     0
```

1.29.14 Repeater Operation

A repeater MUST be used by clients to collect *CA_PROTO_RSRV_IS_UP* messages. Each client host will have one repeater.

Startup

Each client MUST test for presence of repeater on startup, before any access to EPICS hosts is made. This check is made by attempting to bind to `CA_REPEATER_PORT`. If binding fails, the client may assume the repeater is already running and may attempt to register. This is done by sending `CA_REPEATER_REGISTER` datagram to `CA_REPEATER_PORT`. If repeater is already active, it will respond with `CA_REPEATER_CONFIRM` datagram back to client. At this point the registration is complete, and the repeater will begin forwarding messages to the client.

If binding succeeds, then this client process MUST either close the bound socket (and report at error) or begin functioning as a repeater.

If an error is encountered with sending `CA_REPEATER_REGISTER`, then the binding test SHOULD be repeated after a short timeout (1 second is RECOMMENDED).

Client detection

The repeater SHOULD test to see if its clients exist by periodically attempting to bind to their ports. If unsuccessful when attempting to bind to the client's port, then the repeater concludes that the client no longer exists. A technique using connected UDP sockets and ICMP destination unreachable MAY also used. If a client is determined to no longer be present then the repeater un-registers that client and no longer sends messages to it.

Operation

Each message the repeater receives MUST be forwarded to all registered clients.

Shutdown

Repeater should not shutdown on its own, if it does, there should be no active clients registered with it.

1.29.15 Searching Strategy

This section describes one possible strategy for handling *CA_PROTO_SEARCH* messages by a CA client. It is designed to limit the maximum rate at which search messages are sent to avoid overwhelming servers.

For each outstanding search request the following information is kept.

```
struct searchPV {
    const char *pvname;
    epicsTimeStamp nextSend;
    double intervalMult;
};
```

A priority queue should be maintained which is sorted in order of increasing `nextSend`.

When a new search request is made, a new `searchPV` is added to the queue with `initialMult` at a minimum (eg. 0.05 sec.) and `nextSend` at the present time plus `nextSend`.

When a search request is canceled it should be removed from the queue.

A task should run whenever the first entry expires (nextSend before the present time). This task should extract some expired entries up to a maximum limit (eg. enough for 4 UDP packets).

Search messages are then sent for these entries and their intervalMult is increased (eg. doubled), their nextSend is set to the present time plus nextSend, and they are re-added to the queue.

The task should then wait for the minimum search interval (eg. 0.05 sec.) before checking the queue again. This prevents a flood of search messages.

The combination of the minimum interval between sending search messages, and the limit on the maximum number of messages sent in each interval, acts to limit to total network bandwidth consumed by searches.

1.29.16 ECA Error/Status Codes

This section covers return codes and exceptions that can occur during CA command processing. In general, exceptions will be used to report various events to the application. Return codes are predefined values for conditions that can occur, where as exceptions are actually reported. Apart from exceptions that occur on server or due to network transport, additional error conditions may be reported on the client side as local exceptions.

Return codes are represented as UINT16. The 3 least significant bits indicate severity, remaining 13 bits are return code ID.

Return codes are communicated in the protocol by the CA_PROTO_READ_NOTIFY, CA_PROTO_WRITE_NOTIFY, monitor subscription responses, and the CA_PROTO_ERROR responses.

Severity codes

Code	Value	Description
CA_K_SUCCESS	1	Successful (not an error)
CA_K_WARNING	0	Not successful
CA_K_INFO	3	Informational (not an error)
CA_K_ERROR	2	Recoverable failure
CA_K_SEVERE	4	None recoverable failure

Presently defined error conditions

Code	Severity	ID	Value	Description
ECA_NORMAL	CA_K_SUCCESS	0	0x001	Normal successful completion
ECA_ALLOCMEM	CA_K_WARNING	6	0x030	Unable to allocate additional dynamic memory
ECA_TOLARGE	CA_K_WARNING	9	0x048	The requested data transfer is greater than available memory or EPICS_CA
ECA_TIMEOUT	CA_K_WARNING	10	0x050	User specified timeout on IO operation expired
ECA_BADTYPE	CA_K_ERROR	14	0x072	The data type specified is invalid
ECA_INTERNAL	CA_K_FATAL	17	0x08e	Channel Access Internal Failure
ECA_DBLCLFAIL	CA_K_WARNING	18	0x090	The requested local DB operation failed
ECA_GETFAIL	CA_K_WARNING	19	0x098	Channel read request failed
ECA_PUTFAIL	CA_K_WARNING	20	0x0a0	Channel write request failed
ECA_BADCOUNT	CA_K_WARNING	22	0x0b0	Invalid element count requested
ECA_BADSTR	CA_K_ERROR	23	0x0ba	Invalid string
ECA_DISCONN	CA_K_WARNING	24	0x0c0	Virtual circuit disconnect
ECA_EVDISALLOW	CA_K_ERROR	26	0x0d2	Request inappropriate within subscription (monitor) update callback
ECA_BADMONID	CA_K_ERROR	30	0x0f2	Bad event subscription (monitor) identifier
ECA_BADMASK	CA_K_ERROR	41	0x14a	Invalid event selection mask
ECA_IODONE	CA_K_INFO	42	0x153	IO operations have completed

cont

Table 4 – continued from previous page

Code	Severity	ID	Value	Description
ECA_IOINPROGRESS	CA_K_INFO	43	0x15b	IO operations are in progress
ECA_BADSYNCGRP	CA_K_ERROR	44	0x162	Invalid synchronous group identifier
ECA_PUTCBINPROG	CA_K_ERROR	45	0x16a	Put callback timed out
ECA_NORDACCESS	CA_K_WARNING	46	0x170	Read access denied
ECA_NOWTACCESS	CA_K_WARNING	47	0x178	Write access denied
ECA_ANACHRONISM	CA_K_ERROR	48	0x182	Requested feature is no longer supported
ECA_NOSEARCHADDR	CA_K_WARNING	49	0x188	Empty PV search address list
ECA_NOCONVERT	CA_K_WARNING	50	0x190	No reasonable data conversion between client and server types
ECA_BADCHID	CA_K_ERROR	51	0x19a	Invalid channel identifier
ECA_BADFUNCPTR	CA_K_ERROR	52	0x1a2	Invalid function pointer
ECA_ISATTACHED	CA_K_WARNING	53	0x1a8	Thread is already attached to a client context
ECA_UNAVAILINSERV	CA_K_WARNING	54	0x1b0	Not supported by attached service
ECA_CHANDESTROY	CA_K_WARNING	55	0x1b8	User destroyed channel
ECA_BADPRIORITY	CA_K_ERROR	56	0x1c2	Invalid channel priority
ECA_NOTTHREADED	CA_K_ERROR	57	0x1ca	Preemptive callback not enabled - additional threads may not join context
ECA_16KARRAYCLIENT	CA_K_WARNING	58	0x1d0	Client's protocol revision does not support transfers exceeding 16k bytes
ECA_CONNSEQTMO	CA_K_WARNING	59	0x1d9	Virtual circuit connection sequence aborted
ECA_UNRESPTMO	CA_K_WARNING	60	0x1e0	?

Historical error conditions. Servers and clients SHOULD NOT send these codes, but MAY receive them.

Code	Severity	ID	Value	Description
ECA_MAXIOC	CA_K_ERROR	1	0x00a	Maximum simultaneous IOC connections exceeded
ECA_UKNHOST	CA_K_ERROR	2	0x012	Unknown internet host
ECA_UKNSERV	CA_K_ERROR	3	0x01a	Unknown internet service
ECA_SOCK	CA_K_ERROR	4	0x022	Unable to allocate a new socket
ECA_CONN	CA_K_WARNING	5	0x028	Unable to connect to internet host or service
ECA_UKNCHAN	CA_K_WARNING	7	0x038	Unknown IO channel
ECA_UKNFIELD	CA_K_WARNING	8	0x040	Record field specified inappropriate for channel specified
ECA_NOSUPPORT	CA_K_WARNING	11	0x058	Sorry, that feature is planned but not supported at this time
ECA_STRTOBIG	CA_K_WARNING	12	0x060	The supplied string is unusually large
ECA_DISCONNCHII	CA_K_ERROR	13	0x06a	The request was ignored because the specified channel is disconnected
ECA_CHIDNOTFND	CA_K_INFO	15	0x07b	Remote Channel not found
ECA_CHIDRETRY	CA_K_INFO	16	0x083	Unable to locate all user specified channels
ECA_DBLCHNL	CA_K_WARNING	25	0x0c8	Identical process variable name on multiple servers
ECA_ADDFAIL	CA_K_WARNING	21	0x0a8	Channel subscription request failed
ECA_BUILDGET	CA_K_WARNING	27	0x0d8	Database value get for that channel failed during channel search
ECA_NEEDSFP	CA_K_WARNING	28	0x0e0	Unable to initialize without the vxWorks VX_FP_TASK task option set
ECA_OVEVFAIL	CA_K_WARNING	29	0x0e8	Event queue overflow has prevented first pass event after event add
ECA_NEWADDR	CA_K_WARNING	31	0x0f8	Remote channel has new network address
ECA_NEWCONN	CA_K_INFO	32	0x103	New or resumed network connection
ECA_NOCACTX	CA_K_WARNING	33	0x108	Specified task isn't a member of a CA context
ECA_DEFUNCT	CA_K_FATAL	34	00x116	Attempt to use defunct CA feature failed
ECA_EMPTYSTR	CA_K_WARNING	35	0x118	The supplied string is empty
ECA_NOREPEATER	CA_K_WARNING	36	0x120	Unable to spawn the CA repeater thread- auto reconnect will fail
ECA_NOCHANMSG	CA_K_WARNING	37	0x0128	No channel id match for search reply- search reply ignored
ECA_DLCKREST	CA_K_WARNING	38	0x130	Reseting dead connection- will try to reconnect
ECA_SERVBEHIND	CA_K_WARNING	39	0x138	Server (IOC) has fallen behind or is not responding- still waiting
ECA_NOCAST	CA_K_WARNING	40	0x140	No internet interface with broadcast available

1.29.17 Example conversation

This is example conversation between client and server. Client first establishes TCP connection to the server and immediately requests creation of a channel. After server acknowledges channel creation, client reads the value of the channel twice. First as a single string value and second as a DBR_GR_INT16 type. After the response to both queries has been received, the channel is destroyed.

```
Client to Server
CA_PROTO_VERSION (handshake)
00 00  00 00  00 00  00 0b  00 00 00 00  00 00 00 00
    0      0      0     11            0            0
CA_PROTO_CLIENT_NAME (handshake)
00 14  00 08  00 00  00 00  00 00 00 00  00 00 00 00  61 70 75 63 65 6c 6a 00
   20      8      8      0            0            0   a  p  u  c  e  l  j \0
CA_PROTO_HOST_NAME (handshake)
00 15  00 08  00 00  00 00  00 00 00 00  00 00 00 00  63 73 6c 30 36 00 00 00
   21      8      0      0            0            0   c  s  l  0  6 \0 \0 \0
```

(continues on next page)

(continued from previous page)

```
CA_PROTO_CREATE_CHAN (request)
00 12  00 18  00 00  00 00  00 00 00 01  00 00 00 0b
   18     24      0      0           1           11
61 70 75 63 65 6c 6a 3a  61 69 45 78 61 6d 70 6c 65 31 00 00 00 00 00 00
 a  p  u  c  e  l  j  :   a  i  E  x  a  m  p  l  e  1 \0 \0 \0 \0 \0 \0

Server to Client
CA_PROTO_ACCESS_RIGHTS (handshake)
00 16  00 00  00 00  00 00  00 00 00 01  00 00 00 03
   22      0      0      0           1           3
CA_PROTO_CREATE_CHAN (response)
00 12  00 00  00 06  00 01  00 00 00 01  00 00 00 04
   18      0      6      1           1           4
|
Client to Server
CA_PROTO_READ_NOTIFY (request)
00 0f  00 00  00 00  00 01  00 00 00 04  00 00 00 01
   15      0      0      1           4           1
CA_PROTO_READ_NOTIFY (request)
00 0f  00 00  00 16  00 01  00 00 00 04  00 00 00 02
   15      0     22      1           4          02

Server to Client
CA_PROTO_READ_NOTIFY (response)
00 0f  00 08  00 00  00 01  00 00 00 01  00 00 00 01  30 00 00 00 00 06 00 01
   15      8      0      1           1           1   0
CA_PROTO_READ_NOTIFY (response)
00 0f  00 20  00 16  00 01  00 00 00 01  00 00 00 02
   15     32     22      1           1          02
00 05  00 02  43 6f 75 6e 74 73 00 00  00 0a  00 00
    5      2   C  o  u  n  t  s \0 \0     10      0
00 08  00 06  00 04  00 02  00 00  00 00  00 00 00 00
    8      6      4      2      0      0      0     0

Client to Server
CA_PROTO_CLEAR_CHANNEL (request)
00 0c  00 00  00 00  00 00  00 00 00 04  00 00 00 01
   12      0      0      0           4           1

Server to Client
CA_PROTO_CLEAR_CHANNEL (response)
00 0c  00 00  00 00  00 00  00 00 00 04  00 00 00 01
   12      0      0      0           4           1
```

1.29.18 Glossary of Terms

IOC
Input/Output Controller.

PV
Process variable.

Virtual circuit
Reusable TCP connection between client and server, through which all PVs hosted by the server can be conveyed to the client.

1.29.19 References

ID	Author	Reference	Revision	Date	Publisher
1	Jeffrey O. Hill	Channel Access Reference Manual	R3.14	2003	
2		Java Channel Access	2.0.1	2003	
3	Bradner, S.	RFC 2119: Key words for use in RFCs to Indicate Requirement Levels		1997-03	

1.30 IOC Initialization

Tags: advanced

Table of Contents

1.30.1 Overview - Environments requiring a main program

If a main program is required (most likely on all environments except vxWorks and RTEMS), then initialization is performed by statements residing in startup scripts which are executed by iocsh. An example main program is:

```
int main(int argc,char  *argv[])
{
   if (argc >= 2) {
       iocsh(argv[1]);
       epicsThreadSleep(.2);
   }
   iocsh(NULL);
   epicsExit(0)
   return 0;
}
```

The first call to iocsh executes commands from the startup script filename which must be passed as an argument to the program. The second call to iocsh with a NULL argument puts iocsh into interactive mode. This allows the user to issue the commands described in the chapter on "IOC Test Facilities" as well as some additional commands like help.

The command file passed is usually called the startup script, and contains statements like these:

```
< envPaths
cd ${TOP}
dbLoadDatabase "dbd/appname.dbd"
appname_registerRecordDeviceDriver pdbbase
dbLoadRecords "db/file.db", "macro=value"
cd ${TOP}/iocBoot/${IOC}
iocInit
```

The envPaths file is automatically generated in the IOC's boot directory and defines several environment variables that are useful later in the startup script. The definitions shown below are always provided; additional entries will be created for each support module referenced in the application's configure/RELEASE file:

```
epicsEnvSet("ARCH","linux-x86")
epicsEnvSet("IOC","iocname")
epicsEnvSet("TOP","/path/to/application")
epicsEnvSet("EPICS_BASE","/path/to/base")
```

1.30.2 Overview - vxWorks

After vxWorks is loaded at IOC boot time, commands like the following, normally placed in the vxWorks startup script, are issued to load and initialize the application code:

```
# Many vxWorks board support packages need the following:
#cd <full path to IOC boot directory>
< cdCommands
cd topbin
ld 0,0, "appname.munch"

cd top
dbLoadDatabase "dbd/appname.dbd"
appname_registerRecordDeviceDriver pdbbase
dbLoadRecords "db/file.db", "macro=value"

cd startup
iocInit
```

The cdCommands script is automatically generated in the IOC boot directory and defines several vxWorks global variables that allow cd commands to various locations, and also sets several environment variables. The definitions shown below are always provided; additional entries will be created for each support module referenced in the application's configure/RELEASE file:

```
startup = "/path/to/application/iocBoot/iocname"
putenv "ARCH=vxWorks-68040"
putenv "IOC=iocname"
top = "/path/to/application"
putenv "TOP=/path/to/application"
topbin = "/path/to/application/bin/vxWorks-68040"
epics_base = "/path/to/base"
putenv "EPICS_BASE=/path/to/base"
epics_basebin = "/path/to/base/bin/vxWorks-68040"
```

The **ld** command in the startup script loads EPICS core, the record, device and driver support the IOC needs, and any application specific modules that have been linked into it.

dbLoadDatabase loads database definition files describing the record/device/driver support used by the application..

dbLoadRecords loads record instance definitions.

iocInit initializes the various epics components and starts the IOC running.

1.30.3 Overview - RTEMS

RTEMS applications can start up in many different ways depending on the board-support package for a particular piece of hardware. Systems which use the Cexp package can be treated much like vxWorks. Other systems first read initialization parameters from non-volatile memory or from a BOOTP/DHCP server. The exact mechanism depends upon the BSP. TFTP or NFS filesystems are then mounted and the IOC shell is used to read commands from a startup script. The location of this startup script is specified by a initialization parameter. This script is often similar or identical to the one used with vxWorks. The RTEMS startup code calls

```
epicsRtemsInitPreSetBootConfigFromNVRAM(struct rtems_bsdnet_config  *);
```

just before setting the initialization parameters from non-volatile memory, and

```
epicsRtemsInitPostSetBootConfigFromNVRAM(struct rtems_bsdnet_config  *);
```

just after setting the initialization parameters. An application may provide either or both of these routines to perform any custom initialization required. These function prototypes and some useful external variable declarations can be found in the header file epicsRtemsInitHooks.h

1.30.4 IOC Initialization

An IOC is normally started with the **iocInit** command as shown in the startup scripts above, which is actually implemented in two distinct parts. The first part can be run separately as the iocBuild command, which puts the IOC into a quiescent state without allowing the various internal threads it starts to actually run. From this state the second command iocRun can be used to bring it online very quickly. A running IOC can be quiesced using the iocPause command, which freezes all internal operations; at this point the iocRun command can restart it from where it left off, or the IOC can be shut down (exit the program, or reboot on vxWorks/RTEMS). Most device support and drivers have not yet been written with the possibility of pausing an IOC in mind though, so this feature may not be safe to use on an IOC which talks to external devices or software.

IOC initialization using the iocBuild and iocRun commands then consists of the following steps:

Configure Main Thread

Provided the IOC has not already been initialized, initHookAtIocBuild is announced first.

The main thread's epicsThreadIsOkToBlock flag is set, the message "Starting iocInit" is logged and epicsSignalInstallSigHupIgnore called, which on Unix architectures prevents the process from shutting down if it later receives a HUP signal.

At this point, initHookAtBeginning is announced.

General Purpose Modules

Calls coreRelease which prints a message showing which version of iocCore is being run.

Calls taskwdInit to start the task watchdog. This accepts requests to watch other tasks. It runs periodically and checks to see if any of the tasks is suspended. If so it issues an error message, and can also invoke callback routines registered by the task itself or by other software that is interested in the state of the IOC. See "Task Watchdog" for details.

Starts the general purpose callback tasks by calling callbackInit. Three tasks are started at different scheduling priorities.

initHookAfterCallbackInit is announced.

Channel Access Links

Calls dbCaLinkInit. The initializes the module that handles database channel access links, but does not allow its task to run yet.

initHookAfterCaLinkInit is announced.

Driver Support

initDrvSup locates each device driver entry table and calls the init routine of each driver.

initHookAfterInitDrvSup is announced.

Record Support

initRecSup locates each record support entry table and calls the init routine for each record type.

initHookAfterInitRecSup is announced.

Device Support

initDevSup locates each device support entry table and calls its init routine specifying that this is the initial call.

initHookAfterInitDevSup is announced.

Database Records

initDatabase is called which makes three passes over the database performing the following functions:

1. Initializes the fields RSET, RDES, MLOK, MLIS, PACT and DSET for each record.

   Calls record support's init_record (first pass).

2. Convert each PV_LINK into a DB_LINK or CA_LINK

   Calls any extended device support's add_record routine.

3. Calls record support's init_record (second pass).

Finally it registers an epicsAtExit routine to shut down the database when the IOC application exits.

Next dbLockInitRecords is called to create the lock sets.

Then dbBkptInit is run to initialize the database debugging module.

initHookAfterInitDatabase is announced.

Device Support again

initDevSup locates each device support entry table and calls its init routine specifying that this is the final call.

initHookAfterFinishDevSup is announced.

Scanning and Access Security

The periodic, event, and I/O event scanners are initialized by calling scanInit, but the scan threads created are not allowed to process any records yet.

A call to asInit initailizes access security. If this reports failure, the IOC initialization is aborted.

dbProcessNotifyInit initializes support for process notification.

After a short delay to allow settling, initHookAfterScanInit is announced.

Initial Processing

initialProcess processes all records that have PINI set to YES.

initHookAfterInitialProcess is announced.

Channel Access Server

The Channel Access server is started by calling rsrv_init, but its tasks are not allowed to run so it does not announce its presence to the network yet.

initHookAfterCaServerInit is announced.

At this point, the IOC has been fully initialized but is still quiescent. initHookAfterIocBuilt is announced. If started using iocBuild this command completes here.

Enable Record Processing

If the iocRun command is used to bring the IOC out of its initial quiescent state, it starts here.

initHookAtIocRun is announced.

The routines scanRun and dbCaRun are called in turn to enable their associated tasks and set the global variable interruptAccept to TRUE (this now happens inside scanRun). Until this is set all I/O interrupts should have been ignored.

initHookAfterDatabaseRunning is announced. If the iocRun command (or iocInit) is being executed for the first time, initHookAfterInterruptAccept is announced.

Enable CA Server

The Channel Access server tasks are allowed to run by calling rsrv_run.

initHookAfterCaServerRunning is announced. If the IOC is starting for the first time, initHookAtEnd is announced.

A command completion message is logged, and initHookAfterIocRunning is announced.

1.30.5 Pausing an IOC

The command iocPause brings a running IOC to a quiescent state with all record processing frozen (other than possibly the completion of asynchronous I/O operations). A paused IOC may be able to be restarted using the iocRun command, but whether it will fully recover or not can depend on how long it has been quiescent and the status of any device drivers which have been running. The operations which make up the pause operation are as follows:

1. initHookAtIocPause is announced.
2. The Channel Access Server tasks are paused by calling rsrv_pause
3. initHookAfterCaServerPaused is announced.
4. The routines dbCaPause and scanPause are called to pause their associated tasks and set the global variable interruptAccept to FALSE.
5. initHookAfterDatabasePaused is announced.
6. After logging a pause message, initHookAfterIocPaused is announced.

1.30.6 Changing iocCore fixed limits

The following commands can be issued after iocCore is loaded to change iocCore fixed limits. The commands should be given before any dbLoadDatabase commands.

```
callbackSetQueueSize(size)
dbPvdTableSize(size)
scanOnceSetQueueSize(size)
errlogInit(buffersize)
errlogInit2(buffersize, maxMessageSize)
```

callbackSetQueueSize

Requests for the general purpose callback tasks are placed in a ring buffer. This command can be used to set the size for the ring buffers. The default is 2000. A message is issued when a ring buffer overflows. It should rarely be necessary to override this default. Normally the ring buffer overflow messages appear when a callback task fails.

dbPvdTableSize

Record instance names are stored in a process variable directory, which is a hash table. The default number of hash entries is 512. dbPvdTableSize can be called to change the size. It must be called before any dbLoad commands and must be a power of 2 between 256 and 65536. If an IOC contains very large databases (several thousand records) then a larger hash table size speeds up searches for records.

scanOnceSetQueueSize

scanOnce requests are placed in a ring buffer. This command can be used to set the size for the ring buffer. The default is 1000. It should rarely be necessary to override this default. Normally the ring buffer overflow messages appear when the scanOnce task fails.

errlogInit or errlogInit2

These commands can increase (but not decrease) the default buffer and maximum message sizes for the errlog message queue. The default buffer size is 1280 bytes, the maximum message size defaults to 256 bytes.

1.30.7 initHooks

The inithooks facility allows application functions to be called at various states during ioc initialization. The states are defined in initHooks.h, which contains the following definitions:

```
typedef enum {
    initHookAtIocBuild = 0,         / * Start of iocBuild/iocInit commands  */
    initHookAtBeginning,
    initHookAfterCallbackInit,
    initHookAfterCaLinkInit,
    initHookAfterInitDrvSup,
    initHookAfterInitRecSup,
    initHookAfterInitDevSup,
    initHookAfterInitDatabase,
    initHookAfterFinishDevSup,
    initHookAfterScanInit,
    initHookAfterInitialProcess,
    initHookAfterCaServerInit,
    initHookAfterIocBuilt,          / * End of iocBuild command  */

    initHookAtIocRun,               / * Start of iocRun command  */
    initHookAfterDatabaseRunning,
    initHookAfterCaServerRunning,
    initHookAfterIocRunning,        / * End of iocRun/iocInit commands  */

    initHookAtIocPause,             / * Start of iocPause command  */
    initHookAfterCaServerPaused,
    initHookAfterDatabasePaused,
    initHookAfterIocPaused,         / * End of iocPause command  */

/ * Deprecated states, provided for backwards compatibility.
  * These states are announced at the same point they were before,
  * but will not be repeated if the IOC gets paused and restarted.
  */
    initHookAfterInterruptAccept,   / * After initHookAfterDatabaseRunning  */
    initHookAtEnd,                  / * Before initHookAfterIocRunning  */
}initHookState;

typedef void ( *initHookFunction)(initHookState state);
int initHookRegister(initHookFunction func);
const char  *initHookName(int state);
```

Any functions that are registered before iocInit reaches the desired state will be called when it reaches that state. The initHookName function returns a static string representation of the state passed into it which is intended for printing. The following skeleton code shows how to use this facility:

```
static initHookFunction myHookFunction;

int myHookInit(void)
{
  return(initHookRegister(myHookFunction));
}

static void myHookFunction(initHookState state)
{
  switch(state) {
    case initHookAfterInitRecSup:
      ...
      break;
    case initHookAfterInterruptAccept:
      ...
      break;
    default:
      break;
  }
}
```

An arbitrary number of functions can be registered.

1.30.8 Environment Variables

Various environment variables are used by iocCore:

```
EPICS_CA_ADDR_LIST
EPICS_CA_AUTO_ADDR_LIST
EPICS_CA_CONN_TMO
EPICS_CAS_BEACON_PERIOD
EPICS_CA_REPEATER_PORT
EPICS_CA_SERVER_PORT
EPICS_CA_MAX_ARRAY_BYTES
EPICS_TS_NTP_INET
EPICS_IOC_LOG_PORT
EPICS_IOC_LOG_INET
```

For an explanation of the EPICS_CA_... and EPICS_CAS_... variables see the EPICS Channel Access Reference Manual. For an explanation of the EPICS_IOC_LOG_... variables see "iocLogClient" (To be added). EPICS_TS_NTP_INET is used only on vxWorks and RTEMS, where it sets the address of the Network Time Protocol server. If it is not defined the IOC uses the boot server as its NTP server.

These variables can be set through iocsh via the epicsEnvSet command, or on vxWorks using putenv. For example:

```
epicsEnvSet("EPICS_CA_CONN_TMO,"10")
```

All epicsEnvSet commands should be issued after iocCore is loaded and before any dbLoad commands.

The following commands can be issued to iocsh:

epicsPrtEnvParams - This shows just the environment variables used by iocCore.

epicsEnvShow - This shows all environment variables on your system.

1.30.9 Initialize Logging

Initialize the logging system. See the chapter on "IOC Error Logging" for details. The following can be used to direct the log client to use a specific host log server.

```
epicsEnvSet("EPICS_IOC_LOG_PORT", "<port>")
epicsEnvSet("EPICS_IOC_LOG_INET", "<inet addr>")
```

These command must be given immediately after iocCore is loaded.

To start logging you must issue the command:

```
iocLogInit
```

1.31 How to Work with the EPICS Repository

Tags: beginner user developer advanced all

This document aims to show to software developers how to get the current EPICS Base code, modify it and publish the changes.

1.31.1 Organization of the EPICS Git Repository

The main EPICS repository is hosted on Launchpad. The source code repository is at https://git.launchpad.net/epics-base.

A mirror of the repository is available on Github: https://github.com/epics-base/epics-base.git Depending on your location one or the the other may be faster.

All current and past EPICS Base versions are in the same repository on different branches.

EPICS 7

The current development branch is "core/master". However this branch only acts as a kind of envelope. The actual EPICS 7 code is divided into modules each of which lives on a separate branch.

To get the latest version do this:

```
git clone --recursive https://git.launchpad.net/epics-base
cd epics-base
git submodule update --remote
```

This requires at least git version 1.8. Older git versions may need a different procedure. If `git clone --recursive` is not supported, do this instead:

```
git clone https://git.launchpad.net/epics-base
cd epics-base
git submodule update --init --reference .
```

If `git submodule update --remote` is not supported, look up the branches of each module in the .gitmodule file, then go into each module directory and check out the relevant branch manually.

Older EPICS Versions

There is a separate branch for each of the older EPICS Base versions: 3.13, 3.14, 3.15, and 3.16. (However there is no 3.12 branch.)

Use these to check out the latest developments of one of the older versions, for example to fix a bug in one of those versions.

```
git clone --branch 3.14 https://git.launchpad.net/epics-base
```

Specific Releases

Individual releases as well as pre-releases and release candidates are tagged like R3.16.1, R7.0.1-pre or R7.0.1-rc1. First clone the relevant branch, then check out the tag, e.g.:

```
git clone --branch 3.14 https://git.launchpad.net/epics-base
cd epics-base
git co R3.14.8
```

1.31.2 Making Changes

Changes should always be made against the head of the relevant branch, not against the release tags.

For bug fixes check out the branch where the bug appears first. The fix will be merged into newer EPICS versions by the core developer team.

For new features better announce your idea on the core-talk@aps.anl.gov mailing list and ask which branch is most appropriate. For revolutionary new features it is probably the EPICS 7 master branch respectively the branch of the submodule as referenced in the .gitmodule file.

For each change create a new branch with a meaningful name.

```
git checkout -b branch-name
```

Then start working on your change. Don't forget to write a test!

Maintaining Compatibility

Build and test your changes on as many systems as possible. Important operating systems are Linux, Windows, OS X, vxWorks and RTEMS.

Keep in mind that in particular vxWorks 5 uses old compiler versions. Do not break working systems with dependencies on new compiler versions. This means for example C++ 11 features.

EPICS up to 3.15 works with vxWorks 5.5 which uses gcc 3.3.2 with a quite old C++ implementation and EPICS 3.16 works with vxWorks 6.3 using gcc 3.4.4. Do not break that!

Testing

All new features must come with automated tests to prove their functionality. This also helps to find out if future changes break existing features.

There are several "test" directories. Choose the one appropriate for the test. Keep in mind that some tests may run before all parts of Base are built. Details vary depending on the EPICS Base version.

EPICS Base comes with a testing framework which allows to run IOCs, set and read/compare values and more.

To add a test, you will typically create a xxxTest.c and probably some records in a xxxTest.db file. (Choose a suitable name.) Also you need to edit the Makefile in the test directory as well as a file with a name like "epicsRun*Tests.c" to include your new test.

Here is a basic example of a test code (xxxTest.c):

```
#include "dbAccess.h"
#include "dbUnitTest.h"
#include "testMain.h"
MAIN(xxxTest) {
    epicsUInt32 value;

    /* Announce how many test will be done, see comments below. */
    testPlan(total_number_of_tests);

    testdbPrepare();

    /* Load your own IOC or one of the provided. */
    /* "dbTestIoc" or "recTestIoc" may be suitable. */
    testdbReadDatabase("recTestIoc.dbd", NULL, NULL);
    recTestIoc_registerRecordDeviceDriver(pdbbase);

    /* Load your records */
    testdbReadDatabase("xxxTest.db", NULL, "MACRO=VALUE");

    /* start up IOC */
    testIocInitOk();

    /* You may structure the test output with your own comments
     * (This does not count as a test.)
     */
    testDiag("##### This text goes to the test log #####");

    /* Set values and check for success. Counts as 1 test.
     * Make sure that DBF type matches your variable
     */
    testdbPutFieldOk("record.FIELD", DBF_ULONG, value);

    /* Get value and compare with expected result. Counts as 1 test.
     * Make sure that DBF type matches your variable
     */
    testdbGetFieldEqual("record.FIELD", DBF_ULONG, value);

    /* Do some arbitrary test. Counts as 1 test. */
    testOk(condition, formatstring, ...);
```

(continues on next page)

(continued from previous page)

```
    /* The same without your own message. Counts as 1 test. */
    testOk1(condition);

    /* Finish */
    testIocShutdownOk();
    testdbCleanup();
    return testDone();
}
```

Your test should run (and succeed) when you execute

```
make runtests
```

1.31.3 Merging Your Work into EPICS Base

When done with your development, do not push it to the main repository (You probably do not have permission to do so anyway). Instead push it to your personal repository on Launchpad.

Creating a Launchpad Account

If you do not have a Launchpad account yet, got to https://launchpad.net/ and click on "register". With a Launchpad account comes the possibility to have personal repositories. You will use these to push your changes. Don't forget to upload your public (*not private!*) ssh key (found in $HOME/.ssh/id_rsa.pub or similar) in order to be able to push to your repository using ssh.

Pushing Your Work to Launchpad

Before pushing your work, you should first pull the latest version and merge it with your changes if necessary.

In your git working directory, create a new "remote" referring to your personal Launchpad repository. Launchpad will create a new repository if necessary. You can use the same repository for multiple projects on EPICS Base as long as you use different branch names.

```
git remote add launchpad git+ssh://username@git.launchpad.net/~username/epics-base
git push launchpad branch-name
```

After that you can go to the Launchpad web page related to that branch (https://code.launchpad.net/~username/epics-base/+git/epics-base/+ref/branch-name) and click the "Propose for merging" link. The core developer team will review your changes any may either merge them or request fixes.

You can push updates on the same branch at any time, even after making a merge request. The updates will automatically be part of the merge request. Do **not** create a new merge request because of an update!

1.32 Documentation contribution guide

1.32.1 For new contributors

"I'm a newcomer, and I'd like to fix an issue."

Contacting another developer

If you have found a small error or missing piece of information and are unable to fix it yourself, you can email Tech-talk at tech-talk@aps.anl.gov. Provide the details of the issue and another member of the community should be able to fix it for you. You can find more information about using Tech-talk at https://epics.anl.gov/tech-talk/.

Creating an issue on GitHub

Finding the GitHub page and signing up

You can suggest a fix by yourself in the epics-docs GitHub repository. If you would like to create a new page or move information between pages, please refer to the style guide later in this page. For instances where you wish to edit a page, follow the GitHub link in the top-right of the page.

Following this link, will take you to the source page. If you don't already have a GitHub account, go to the join GitHub page and follow the instructions.

Creating an issue

At the top of the GitHub page, click on the "Issues" tab. If there are no issues already listed about the problem that you want fixed, click the "New issue" button, give your issue a title and write a brief description, then submit it. If nobody has picked up your issue after several days, email Tech-talk and a developer should pick it up.

1.32.2 Making a contribution

Structure

The various documents are divided by topic, for example:

- Getting started
- Process database
- Modules
- etc.

This documentation also follows the Diátaxis documentation framework. We recommend reading the Diátaxis documentation.

What this means for the EPICS documentation, is that documentation falls into 4 categories:

- Tutorials: for users to learn new concepts
- How-to guides: for users to achieve specific, predefined goals
- Explanations: for users to clarify their understanding of a concept
- References: for users to consult, when looking for specific information

Each of those type of page must be put in that order in the various topics. For example:

- Process database
  - Making an IOC with linked PVs (tutorial)
  - How to find which IOC provides a PV (guide)
  - How to Avoid Copying Arrays with `waveformRecord` (guide)
  - Process Database Concepts (explanation)
  - Record reference (reference)

Tagging the document

To suggest the intended audience, we use a tag mechanism using sphinx-tags extension.

There are `beginner`, `user`, `developer`, `advanced` and `all` roles.

- `beginner` - articles for those who don't know EPICS and want to familiarize with it. That should be mostly articles about installation and basic concepts.
- `user` - articles for those who use EPICS during their work for example as operators, mostly with client applications.
- `developer` - articles for those who develop IOCs, extensions or drivers
- `advanced` - articles for those who want to understand advanced topics including build system, specifications and details of protocols.

To tag the article, add a line specifying labels under the title in the source file. Tags are supported for `.rst` and `.md`.

Example for `.rst`:

```
.. tag::`beginner, user, developer`
```

Example for `.md`:

````
```{tags} tag1, tag2
```
````

Forking the repository

Once logged in and viewing the page on GitHub you wish to edit, click on the pencil icon to the top-right of the content. If this is your first time editing, you will see with a page asking you to fork the repository before being able to edit. Click through the link to do this, and GitHub will create a copy of the entire repository linked to your own account. Feel free to edit any page in this repository. Your changes won't show up in the main repository or the Read the Docs site until you create a pull request.

Local setup and build

"How do I build the epics-doc documentation locally, and how do I serve it locally?" Being able to do this can be of interest to check how your contribution will render before sharing it.

Using poetry

A practical solution to build the epics-docs documentation is to use Poetry. Poetry is a tool for dependency management and packaging in Python. It's reproducible, no matter what your environment is, and multi-platform (it works equally well on Linux, macOS and Windows).

Please follow the Poetry documentation to install it.

Once installed, you can setup, build, and serve the epics-docs documentation in two steps:

1. Clone (with SSH) your epics-docs fork (see *Forking the repository* above) and change directory into it:

```
$ git clone git@github.com:your-user-name/epics-docs.git
$ cd epics-docs
```

2. Install Poetry dependencies (the "setup") and build + serve the epics-docs documentation:

```
$ poetry install
$ poetry run sphinx-autobuild --re-ignore _tags/ . ./_build/html
```

At this point, you can open http://127.0.0.1:8000 in your internet browser and check the generated documentation by yourself.

Using pip

Another solution for local builds is to use pip (ideally in a virtual environment), which comes pre-installed with most modern python installations. From the pip website:

> pip is the package installer for Python. You can use it to install packages from the Python Package Index and other indexes

1. Follow the first step in the *Using poetry* section to locally clone the epics-docs repository.
2. Install the pip dependencies from `requirements-dev.txt`

```
$ python -m venv venv                  # Create a virtual environment for your local␣
↪build
$ . venv/bin/activate                  # Activate it
$ pip install -r requirements-dev.txt
$ sphinx-autobuild . ./_build/html
```

At this point, same as above, you can open http://127.0.0.1:8000 in your internet browser, and check the generated documentation by yourself.

Reference setup and build

You can check for yourself how the project is built here: https://readthedocs.org/projects/epics/builds/, and then click on the last "Passed" build (that is, the last build that succeeded), for example: https://readthedocs.org/projects/epics/builds/21001074/.

Warning: Those commands are oriented to optimize the build in the Read the Docs environment. This might not always be reproducible on your computer without some level of modifications depending on your own local environment, such as your OS, your distribution, your shell interpreter, etc. This way is how Read the Docs builds the documentation, but it isn't always the most practical way (at least not for development/contribution purposes).

Edit and view your changes

Now that you know how to clone, setup, build, and serve the epics-docs documentation, you can edit any `.rst` or `.md` file and check how sphinx will render your contribution.

After running `$ poetry run sphinx-autobuild . ./_build/html`, like described earlier, any modification will update http://127.0.0.1:8000 automatically.

Style guide

This section covers the conventions when writing documentation.

You should write new documentation in Markdown. Some existing documentation might be in reStructuredText (RST), but these pages should progressively be converted to Markdown.

If you're unfamiliar with Markdown, you can look at the Basic Syntax page from the Markdown Guide website.

Another convention used is Semantic line breaks, which increase readability and make editing the source easier: add line breaks after each sentence, after independent clauses (comma, semicolon, colon, dash), and before every relative clause.

A configuration for Vale also exists in the repository, to help you write English documentation. You are *not* required to fix every Vale warning, these are meant as advice.

To see those Vale warnings, install Vale by following the Vale Installation guide, and run `vale path/to/your/file.md`.

Making a pull request

After you are satisfied with your changes, commit and push them to your fork. Keeping separate things in separate commits will make reviewing easier. Finally submit the branch with your commits for review by creating a pull request.

To create a pull request, first click on the "Pull requests" tab at the top of GitHub. From here, click the green "New pull request" button, which should take you to a page comparing the main repository to your fork. You should see any commits you have made listed here. Clicking "Create pull request" will give you the opportunity to give your edits a title and a brief description, before you submit them for review.

At this point, a maintainer of the repository will be able to review your changes to ensure they're sensible and don't break anything. If all is well, they will approve the changes and merge them into the main repo. After the reviewer has merged the pull request, Read the Docs will recompile the page and publish your changes.

1.32.3 Adding dependencies

When adding a dependency, make sure to add them to both Poetry and the requirement files.

For example, if you want to add the `sphinx-foo` extension, first run:

```
poetry add sphinx-foo
```

This command adds that extension to both `pyproject.toml` and `poetry.lock`. It also fetches the latest version and pins it in `pyproject.toml`.

To add that dependency to the requirement files, take the version pinned in `pyproject.toml`, and make sure to pin that same version in the requirements files.

For example, if you see in `pyproject.toml`:

```
sphinx-foo = "^1.1.2"
```

Then you must add to both `requirements.txt` and `requirements-dev.txt`:

```
sphinx-foo==1.1.2
```

1.32.4 Reviewing pull requests

From the point of view of a reviewer.

TODO: how to merge (merge commit (not FF))

TODO: say if pushing to PRs from a reviewer point of view is acceptable

1.33 How to run an EPICS Collaboration Meeting

Tags: user developer advanced

This page is intended for "lessons learned" by sites who have run collaboration meetings, as hints to help future meetings run smoothly.

1.33.1 Organization

The EPICS Council now decides where meetings will be held, so there are usually 2 meetings a year, circulating between the Americas, Europe and the Asia/Pacific region. Future meeting locations that have already been fixed are usually listed here.

Collaboration meetings are usually 3 days long from Tuesday to Thursday or Wednesday to Friday, allowing for training and smaller working groups on the other days of the week. You'll need a big room for the full meeting, and some smaller meeting rooms for any workshops and training sessions you host.

Also somewhere near the refreshment location a room for any exhibitions; past meetings have succeeded in attracting companies who will a small sum to put up a display table at these meetings. Providing large-screen TVs or projectors in the exhibition space allows EPICS-related projects to demonstrate their software.

Recent attendance has been between 80-130 people, but this depends on the number of users from the hosting site and nearby. It can be difficult for government-funded workers (especially from US Department of Energy labs) to get approval for travel to exotic locations though, so expect attendance to vary. Workshops can be around 30 people, but may vary depending on the topic. If someone wants to run a workshop you can ask them for estimates on numbers.

To include some hands-on training you may may provide PCs capable of running a virtual machine for students to use, or ask them to bring their own laptops (which some will do anyway). Training is still possible without these, but would consist of basic lectures and demo's only. You would need to decide what kind of training you want, and organize some people in the community to give it.

Topics for workshops should generally be aligned with interests of the hosts. It will take support on the host side to make sure there is sufficient interest and attendance. Subgroups of developers such as the EPICS Core, CS-Studio and AreaDetector groups may ask to hold a private developers meeting adjacent to the main meeting, which will usually require providing a 10-15 seat meeting room with WiFi for each group for the period of their meeting.

1.33.2 Communications

If possible, start creating a website for the meeting before it is first announced to on tech-talk; some people will want to be able to find out more about the location when they first hear about it, so the announcement should have a link to that website.

Attendees from some countries such as China may have to navigate a quite long approvals process (internal, government and visa) to be able to attend, and may need a letter of invitation from you to get those approvals. Provide enough information so they know who to ask well in advance (3-4 months or more is advisable).

Eventually you'll most probably need online registration of attendees and possibly speakers, as well as to provide information about local hotels, transportation to/from nearby airports, and local tourist agencies for attendees' partners. Arranging a "partner programme" is usually unnecessary.

1.33.3 Facilities

For the main meeting, a hall with LCD projector(s) for computer connection and a PC to display the presentation files. Most people will bring talks in MS PowerPoint, Adobe PDF and/or LibreOffice Impress formats (installing LibreOffice on this machine is advised, but not essential if you give people notice). It saves time and confusion at these international meetings if the PC can be configured with English language settings (Windows menus etc.). Providing a remote control for the presentation program and a laser pointer is helpful.

Some presenters may want to use their own laptops for live demonstrations, or if they're using a less common presentation program (e.g. Apple's Keynote).

In a large hall speakers should use a microphone if a PA is available; a radio-microphone is preferred.

The main hall (and ideally the other meeting rooms too) should have reliable, high-bandwidth WiFi internet access since most people will bring a laptop or notebook PC and want to connect up during the meeting.

Laptop batteries don't last more than a few hours, so there will be demand for power sockets in the hall too. If these are not available at every seat, providing extension leads spread about the room is generally a good idea and is much appreciated by attendees.

Refreshments should be available at the breaks mid-morning and mid-afternoon. If your institution can't afford to fund these itself it is acceptable to charge attendees a fee at registration to pay for them (exhibitors funds are also helpful here). People will expect to pay for their own lunches, and also to pay to attend the conference dinner which is usually held on the evening before the last day.

1.33.4 Agenda

Setting an initial program structure gives presenters an idea of what topics the hosting institution may be particularly interested in, but isn't necessary.

In recent years the hosting site has been responsible for soliciting and collecting speakers names and talk titles; an Indico website can perform much of the clerical work involved automatically, but the talks will still have to be manual scheduled into the Agenda. Individual talks are usually allotted 15 minutes plus 5 minutes for questions, but some topics may be given additional time. Recent meetings have also introduced "Lightning talks" which are 5 minutes long with no question time, and these have proven popular and a good way to cover many topics in a short time.

The community will usually submit a number of submissions of presentations, but the hosts should expect to have to do some additional solicitation to fill out the program. This can (should) be informed by topics the host institute is interested in. Communications to the whole community about submitting talks should be sent to tech-talk.