EPICS Documentation

Apr 07, 2020

Contents

This is the parent project for all EPICS related documentation on read-the-docs.

A good part of the EPICS documentation is dynamically created and hosted by read-the-docs, which acts as a CI system, building and hosting documentation from source or dedicated documentation repositories.

Read-the-docs offers valuable features that we use:

- Keeps complete frozen documentation trees of all released versions under a permalink with the release number.
- Keeps the documentation of the development tree (usually 'master') up-to-date and accessible using 'latest' as release number.
- Switching versions is available through the web browser.

This parent project page is not directly linked from the EPICS web site. It creates the link to the canonical URL https://docs.epics-controls.org - links from the EPICS web site to the documentation should point to the subprojects using the canonical URL, e.g., 'https://docs.epics-controls.org/projects/<project-slug>'

There are two kinds of subprojects:

1. Code repositories: The documentation is created from comments in the source code (Javadoc or Doxygen) and separate documents (PDF, Markdown, restructuredText). Documentation trees of released versions are kept for reference.
2. Documentation repositories: The repositories contain only documentation (like FAQs and How-To-Pages). Usually only one version exists ('latest').

CHAPTER 1

EPICS Overview

1.1 What is EPICS?

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 [1]
- 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.

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.

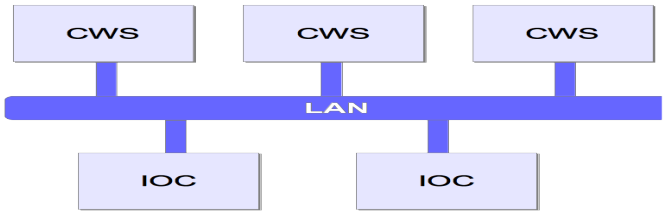


Figure 1. A simple EPICS control system structure.

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.

1.2 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.

1.3 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.

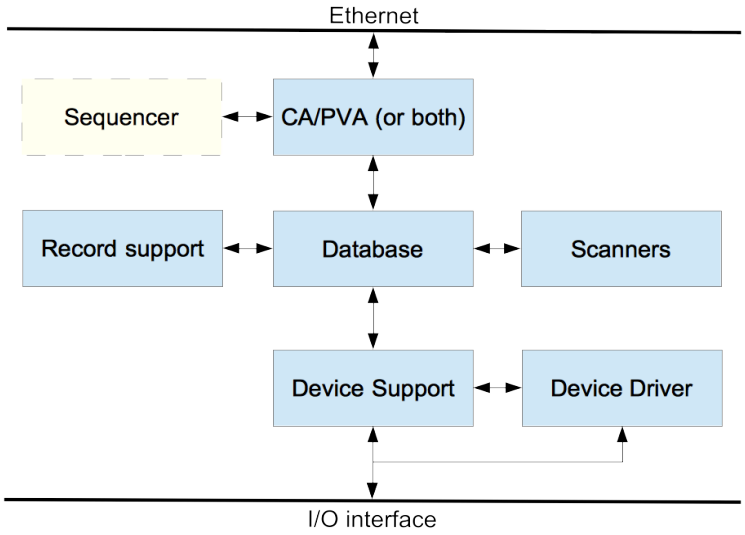


Figure 2. EPICS IOC components.

1.3.1 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
```

(continues on next page)

(continued from previous page)

```
  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).

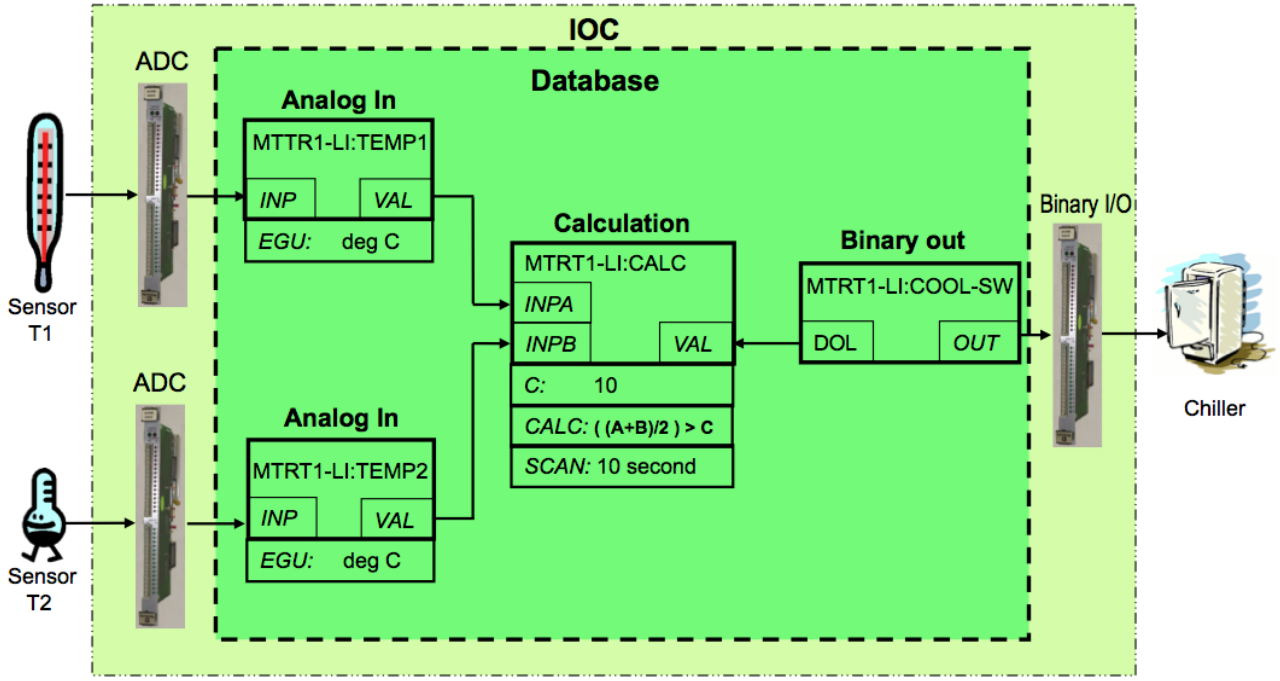


Figure 4. Example of record linking. From [2].

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.

1.3.2 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.

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.

1.3.3 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.

1.4 Network protocols

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

1.4.1 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.

1.4.2 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.

1.5 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.

1.5.1 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.

1.5.2 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.

1.6 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.

1.7 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.

[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.

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

CHAPTER 2

EPICS Process Database Concepts

Table of Contents

2.1 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).

2.2 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.

2.3 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.

2.3.1 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 numeric value specified in units of seconds. 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.

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.3.5 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.

2.3.6 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.

2.3.7 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.

2.3.8 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.

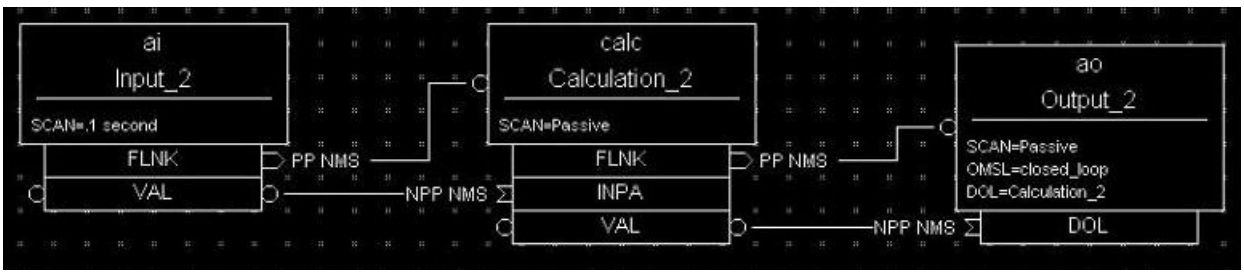


Figure 1. Input Links

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.

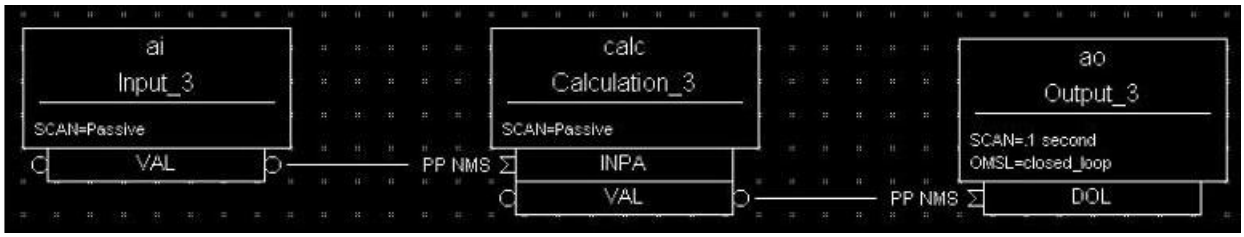


Figure 2

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.

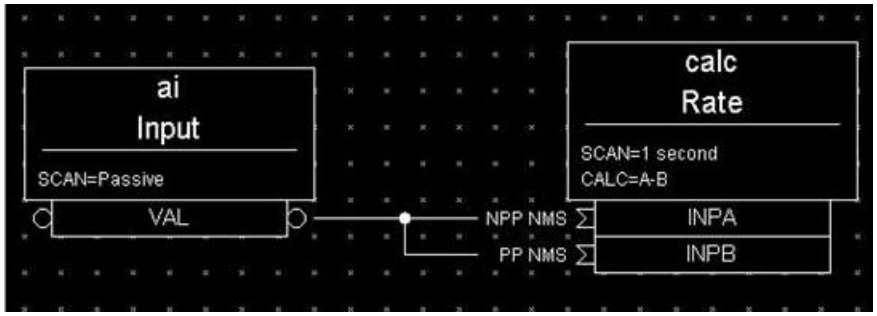


Figure 3

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.

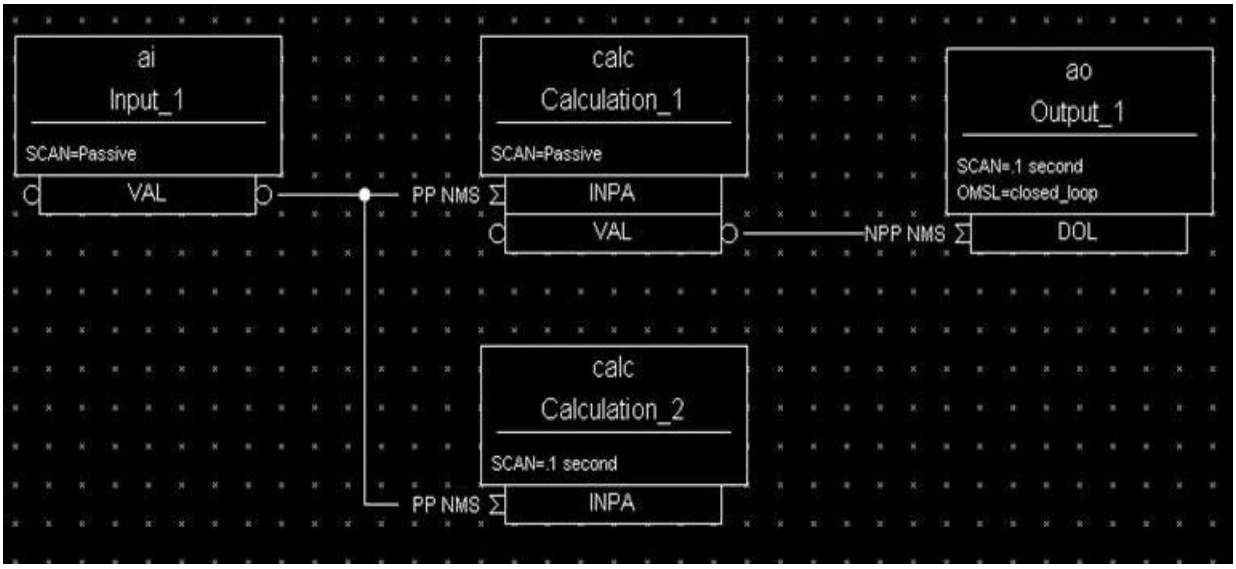


Figure 4

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.

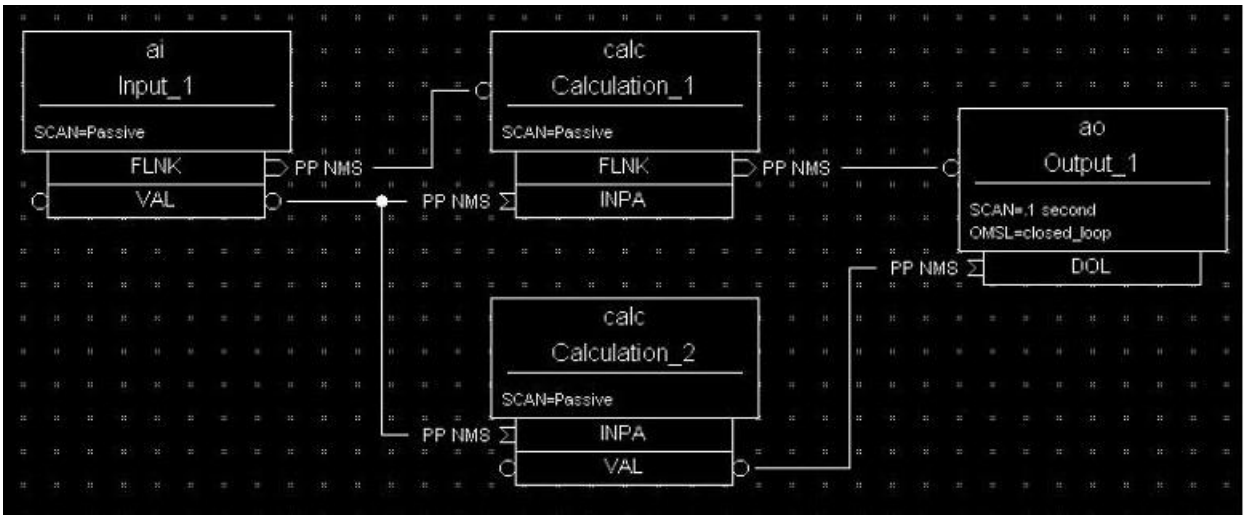


Figure 5

2.3.9 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.

2.3.10 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.

2.3.11 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.

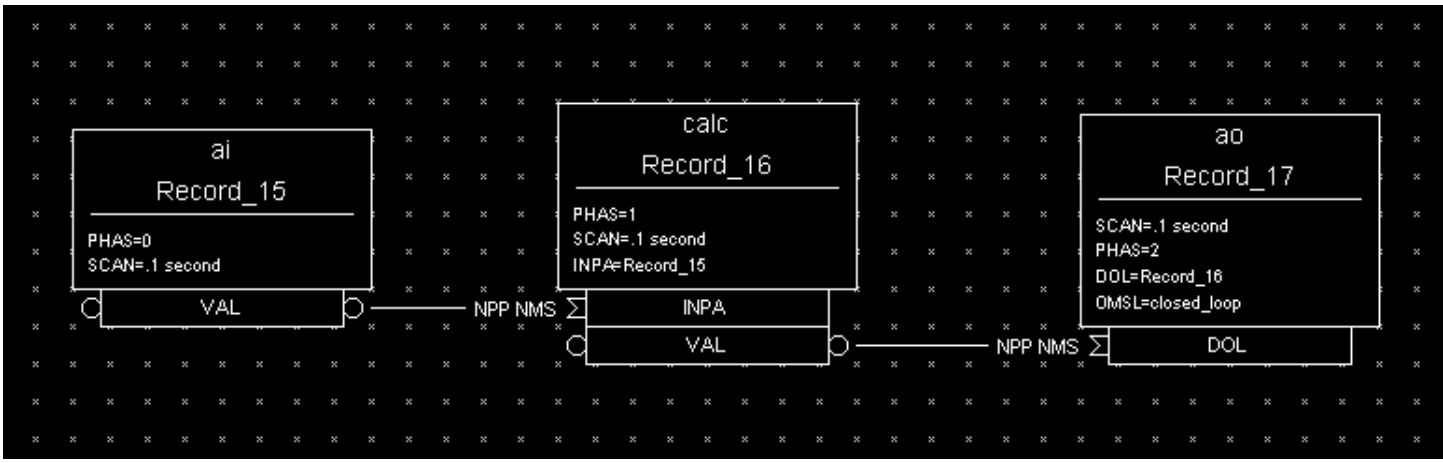


Figure 6

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.

2.3.12 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",
```

(continues on next page)

(continued from previous page)

```
    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.

2.4 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: 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.

2.4.1 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*

2.4.2 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.

2.5 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.

2.5.1 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.

2.5.2 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
- NO CONVERSION
- typeKdegF
- typeKdegC
- typeJdegF
- typeJdegC

If LINEAR 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 that 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.

2.5.3 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.

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

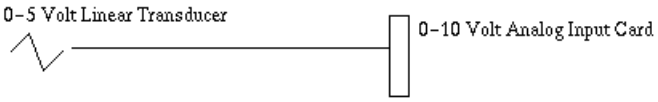
The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = 0 + \frac{\text{measured A/D counts}}{4095} * (175 - 0)$$

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:

$$\text{eng units} = -175 + \frac{\text{measured A/D counts}}{4095} * (175 - (-175))$$

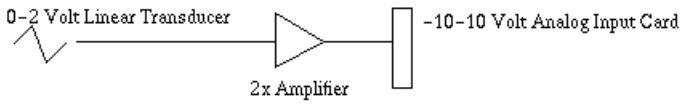
Notice that at low scale the transducer will generate 0 Volts to represent 0 PSI. Because this is half of what the input card accepts, it is input as 2048. Let's plug in the numbers to see the result:

$$-175 + (2048/4095) * (175 - (-175)) = 0$$

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.

2.5.4 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

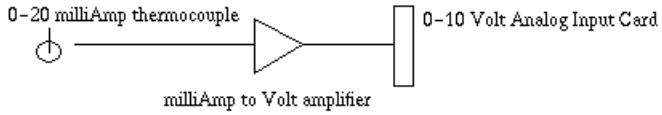
When a new raw value is read, the conversion routine starts from the previously used line segment, compares the raw value start, and either going forward or backward in the table searches the proper segment for this new raw value. Once the proper segment is found, the new engineering units value is the 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

2.6 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.

2.6.1 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.

2.6.2 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.

2.6.3 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.

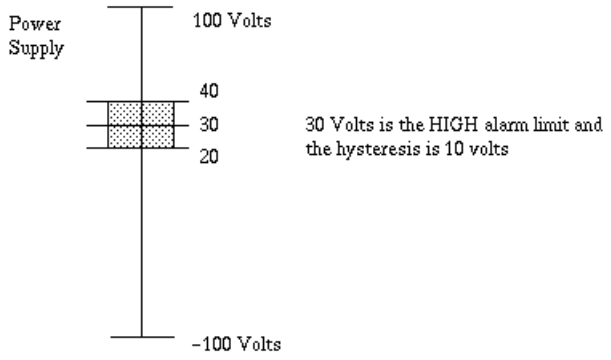
Limit Alarms

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.

2.6.4 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.

2.7 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.

2.7.1 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 that 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.

2.7.2 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.

2.8 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”.

2.8.1 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.

2.8.2 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.

CHAPTER 3

Specifications

Specifications for the different software modules and conventions related to EPICS.

3.1 Channel Access Protocol Specification

Table of Contents

3.1.1 License

3.1.2 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

3.1.3 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 implements. The focus is on versions >= 4.11 of the CA protocol, which 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.

3.1.4 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	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

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.

3.1.5 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.

3.1.6 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[n]	char[]	Array of UBYTE`s. If `[n] is specified, it indicates maximum allowed number of characters in this string including (if neccessary) termination character.
TIMESTAMP	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.

3.1.7 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 the 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 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.

3.1.8 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_V411	Server will send response immediately after establishing a virtual circuit.
< CA_V411	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 <#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 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 identifer.
SearchID		Client allocated Search identifer.

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 <#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 identifer.

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 <#ca-proto-sear ch>‘__, 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 identifer.
SearchID		Client allocated Search identifer.

Table: Table 11. Header

Comments

- Contents of the header are identical to the request.
- SearchID fields are diplicated.
- 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

3.1.9 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 beggining (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

3.1.10 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 <#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 <#ca-proto-event-add>‘__.
Data Count	>= 0	Same value as in corresponding `CA_PROTO_EVENT_ADD <#ca-proto-event-add>‘__.
SID	SID of channel	Same value as in corresponding `CA_PROTO_EVENT_ADD <#ca-proto-event-add>‘__.
SubscriptionID	Subscription ID	Same value as in corresponding `CA_PROTO_EVENT_ADD <#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 <#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 <#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

3.1.11 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
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

Continued on next page

Table 1 – continued from previous page

Name	ID	Meta size	padding	Element size
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;
```

(continues on next page)

(continued from previous page)

```
    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;
    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;
```

(continues on next page)

(continued from previous page)

```
    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.

3.1.12 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 overriden by application.

Port numbers are dependant on protocol versions and are calculated using the folowing 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 the whether 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. Similiar 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. Accss rights are defined as logicaly ORred value 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.

3.1.13 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 messsage 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
```

3.1.14 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, the 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.

3.1.15 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.

3.1.16 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 EPI
ECA_TIMEOUT	CA_K_WARNING	10	0x050	User specified timeout on IO operation expired
ECA_BADTYPE	CA_K_ERROR	14	0x072	The data type specifed 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
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 co
ECA_16KARRAYCLIENT	CA_K_WARNING	58	0x1d0	Client's protocol revision does not support transfers exceeding 16k l
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_DISCONNCHID	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 isnt 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

3.1.17 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 aknowledges 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
```

(continues on next page)

(continued from previous page)

```
   21      8      0      0            0            0   c  s  l  0  6 \0 \0 \0
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
```

3.1.18 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.

3.1.19 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	

3.2 IOC Access Security

Table of Contents

3.2.1 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

3.2.2 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.

3.2.3 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)
        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).

3.3 IOC Initialization

Table of Contents

        * *callbackSetQueueSize*
        * *dbPvdTableSize*
        * *scanOnceSetQueueSize*
        * *errlogInit or errlogInit2*
    - *initHooks*
    - *Environment Variables*
    - *Initialize Logging*

3.3.1 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")
```

3.3.2 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.

3.3.3 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

3.3.4 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.

3.3.5 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.

3.3.6 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.

3.3.7 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,
```

(continues on next page)

(continued from previous page)

```
    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.

3.3.8 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.

3.3.9 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
```

3.4 Build Facility

Table of Contents

Janet Anderson is the author of this chapter.

3.4.1 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.

3.4.2 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.

3.4.3 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>
gnumake clean uninstall
edit configure/RELEASE
```

change the relevant line(s) to point to the new release

```
gnumake
```

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 <osclass> is vxWorks.
- For RTEMS-* targets <osclass> is RTEMS.
- For solaris-* targets <osclass> is solaris.
- For win32-* targets <osclass> is WIN32.
- For linux-* targets <osclass> is Linux.
- For darwin-* targets <osclass> is Darwin.
- For aix-* targets <osclass> 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>
```

3.4.4 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 <dir1> first and then <dir2>. 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
```

3.4.5 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<arch> 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:

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

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

NOTE: 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.

3.4.6 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.

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 Note: Only archive libraries are created.

Shared libraries Note: 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 Note: 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: 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.

item 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 <name1> script being installed from the src directory to the $(INSTALL_LOCATION)/bin/<arch> directories for all target archs of the specified os class <osclass> and the <name2> 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 <arch> 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 form:

RCS += <name> Resource definition script files for all products and libraries.
RCS_<osclass> += <name>

PROD_RCS += <name> Resource definition script files for all products.
PROD_RCS_<osclass> += <name>
PROD_RCS_DEFAULT += <name>

LIB_RCS += <name> Resource definition script files for all libraries.
LIB_RCS_<osclass> += <name>
LIB_RCS_DEFAULT += <name>

<name>_RCS += <name> Resource definition script files for specified product or library.
<name>_RCS_<osclass> += <name>
<name>_RCS_DEFAULT += <name>

result in resource files (*.res files) being created from the specified *.rc resource definition script files and linked into the prods and/or libraries.

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 1

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 2

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

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

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:    (G=GLOBAL)
    D10_init
    20_environment
    30_drivers
    D40_settings
    70_start-ioc
```

```
../st.cmd.d:         (L=LOCAL)
    D10_init
    40_settings
    40_settings~
    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	10_init	L default resets the G default
G	20_environment	
L	30_another-driver	implicit addition, alphabetical sorting
G	30_drivers	
L	40_settings	replacing a default, ignoring backup file
L	70_start-my-ioc	explicit replace

3.4.7 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
PROD_<osclass>	os class specific products to 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_<oscla
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	products to build and install for arch with no PROD_HOST_<osclass> specified
Test products to be built	
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> spec
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 TESTPROD_IO
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 products to build and install for arch with no TESTPROD_HOST_<osclass> s
Test scripts to be built	
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> spe
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_I
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	test scripts to build and install for arch with no TESTSCRIPTS_HOST_<osclass>
Libraries to be built	

Table 3 – continued from previous

LIBRARY	name of library to build and install. The name should NOT include a prefix or exte
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 inclu
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_<o
LIBRARY_HOST	name of library to build and install for host type archs. The name should NOT incl
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_
SHARED_LIBRARIES	build shared libraries? Must be YES or NO
SHRLIB_VERSION	shared library version number
Loadable libraries to be built	
LOADABLE_LIBRARY	name of loadable library to build and install. The name should NOT include a pref
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_<
LOADABLE_LIBRARY_HOST	name of loadable library to build and install for host type archs. The name should
LOADABLE_LIBRARY_HOST_<osclass>	os class specific loadable libraries to build and install for host type archs
LOADABLE_LIBRARY_HOST_DEFAULT	loadable libraries to build and install for host type arch systems with no LOADAB
Combined object files (vxWorks only)	
OBJLIB	name of a combined object file library and corresponding munch file to build and i
OBJLIB_vxWorks	same as OBJLIB
OBJLIB_SRCS	source files to build the OBJLIB
OBJLIB_OBJS	object files to include in OBJLIB
Product and library source files	
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 SRCS_<osclass>
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>
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	source files needed to build a specific PROD or LIBRARY for archs with no <proc
Product and library object files	
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 LIBR
USR_OBJS_DEFAULT	object files, specified without suffix, needed to build PRODs and LIBRARYs for a
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

Table 3 – continued from previous

PROD_OBJS_DEFAULT	object files, specified without suffix, needed to build PRODs for archs with no OB
LIB_OBJS	object files, specified without suffix, for building all LIBRARYs (e.g. LIB_OBJS+
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
<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 LIBR
<name>_OBJS_DEFAULT	object files, without suffix, needed to build a specific PROD or LIBRARY for arch
Product and library R3.13 combined object files	
USR_OBJLIBS	combined object files with filenames that do not have a suffix, needed for building
USR_OBJLIBS_<osclass>	os-specific combined object files with filenames that do not have a suffix for buildi
USR_OBJLIBS_DEFAULT	combined object files with filenames that do not have a suffix, for archs with no U
PROD_OBJLIBS	combined object files with filenames that do not have a suffix, needed for building
PROD_OBJLIBS_<osclass>	os-specific combined object files with filenames that do not have a suffix for buildi
PROD_OBJLIBS_DEFAULT	combined object files with filenames that do not have a suffix, for archs with no PI
LIB_OBJLIBS	combined object files with filenames that do not have a suffix, needed for building
LIB_OBJLIBS_<osclass>	os-specific combined object files with filenames that do not have a suffix for buildi
LIB_OBJLIBS_DEFAULT	combined object files with filenames that do not have a suffix, for archs with no LI
<name>_OBJLIBS	combined object files with filenames that do not have a suffix, needed to build a sp
<name>_OBJLIBS_<osclass>	os specific combined object files with filenames that do not have a suffix, to build a
<name>_OBJLIBS_DEFAULT	combined object files with filenames that do not have a suffix, needed to build a sp
<name>_LDOBJS	combined object files with filenames that do not have a suffix, needed to build a sp
<name>_LDOBJS_<osclass>	os specific combined object files with filenames that do not have a suffix, to build a
<name>_LDOBJS_DEFAULT	combined object files with filenames that do not have a suffix, needed to build a sp
Product and library dependant libraries	
<name>_DIR	directory to search for the specified lib. (For libs listed in all PROD_LIBS, LIB_L
USR_LIBS	load libraries (e.g. Xt X11) for 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_<osc
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> specifie
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
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> 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>_SY
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_<osc
LIB_SYS_LIBS	system libraries to be linked with every library being created

Table 3 – continued from previous

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_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	system libs needed to link every PROD for systems with no SYS_PROD_LIBS_<
Compiler flags	
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
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_EXTENSION
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>_
<name>_INCLUDES	directories, with -I prefix, to search for include files when building a specific objec
<name>_INCLUDES_<T_A>	file specific directories, with -I prefix, to search for include files for a specific targe
<name>_INCLUDES_<osclass>	file specific directories, with -I prefix, to search for include files for a specific os cl
HOST_WARN	Are compiler warning messages desired for host type builds? (YES or NO) (defau
CROSS_WARN	C cross-compiler warning messages desired (YES or NO) (default YES)
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	C++ compiler selection, NORMAL or STRICT (default is STRICT)
Linker options	
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

Table 3 – continued from previous

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 <name>_LDFLAGS_<osclass> specified
STATIC_BUILD	Is static build desired (YES or NO) (default is NO). On win32 if STATIC_BUILD
Header files to be installed	
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	include files to install where no INC_<osclass> is specified
Perl, csh, tcl etc. script installation	
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> speci
SCRIPTS_HOST	scripts to install for host type archs. T
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> speci
TCLLIBNAME	list of tcl scripts to install into $(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.ob
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_<osc
OBJS_HOST	object files to build and install for host type archs. T
OBJS_HOST_<osclass>	os class specific object files to build and install for host type archs
OBJS_HOST_DEFAULT	object files to build and install for host type arch systems with no OBJS_HOST_<
Documentation	
DOCS	text files to be installed into the $(INSTALL_DIR)/doc directory
HTMLS_DIR	name install Hypertext directory name i.e. $(INSTALL_DIR)/html/$(HTMLS_DI
HTMLS	hypertext files to be installed into the $(INSTALL_DIR)/html/$(HTMLS_DIR) dir
TEMPLATES_DIR	template directory to be created as $(INSTALL_DIR)/templates/$(TEMPLATE_D
TEMPLATES	template files to be installed into $(TEMPLATE_DIR)
Database Definition files	
DBD	database definition files to be installed or created and installed into $(INSTALL_D
DBDINC	names, without suffix, of menus or record database definitions and headers to be in
USR_DBDFLAGS	optional flags for dbExpand. Currently only include path (-I <path>) and macro su
DBD_INSTALLS	files from specified directory to install into $(INSTALL_DBD) (e.g. DBD_INSTA
Database Files	
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
USR_DBFLAGS	optional flags for msi (EPICS Macro Substitution Tool)
Options for other programs	

Table 3 – continued from previous

YACCOPT	yacc options
LEXOPT	lex options
SNCFLAGS	state notation language, snc, options
<name>_SNCFLAGS	product specific state notation language options
E2DB_FLAGS	e2db options
SCH2EDIF_FLAGS	sch2edif options
RANLIBFLAGS	ranlib options
USR_ARFLAGS	ar options
Facilities for building Java programs	
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	javah tool options
Facilities for Windows 95/NT resource (.rc) files	
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 typ
RCS_DEFAULT	resource files needed to build every PROD and LIBRARY for ioc type arch system
<name>_RCS	resource files needed to build a specific PROD or LIBRARY
<name>_RCS_<osclass>	os specific resource files to build a specific PROD or LIBRARY
<name>_RCS_DEFAULT	resource files needed to build a specific PROD or LIBRARY for ioc type arch syst
Assemblies	
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	patterns for snippet files (searched from all source directories) needed to build a sp
Other definitions:	
USR_VPATH	list of directories
BIN_INSTALLS	files from specified directories to be installed into $(INSTALL_BIN) (e.g. BIN_IN
BIN_INSTALLS_<osclass>	os class specific files from specified directories to be installed into $(INSTALL_BI
BIN_INSTALLS_DEFAULT	files from specified directories to be installed into $(INSTALL_BIN) for target arc
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_LI
LIB_INSTALLS_DEFAULT	files from specified directories to be installed into $(INSTALL_LIB) for target arc
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 MODULE
GENVERSIONDEFAULT	The default version string written into the generated header if no VCS system is in

3.4.8 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 <osclass> and DEFAULT options.

CONFIG_APP_INCLUDE

Definitions to generate include, bin, lib, perl module, db, and dbd directory definitions for RELEASE <top>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 <tops> 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/<iocname> 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. The 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).

3.4.9 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.

3.4.10 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

CHAPTER 4

Base

User Manuals and API documentation for the different software modules that comprise EPICS Base.

4.1 How to Work with the EPICS Repository

This document aims to show to software developers how to get the current EPICS Base code, modify it and publish the changes.

4.1.1 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
```

4.1.2 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, ...);

    /* The same without your own message. Counts as 1 test. */
    testOk1(condition);
```

(continues on next page)

(continued from previous page)

```
    /* Finish */
    testIocShutdownOk();
    testdbCleanup();
    return testDone();
}
```

Your test should run (and succeed) when you execute

```
make runtests
```

4.1.3 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!

CHAPTER 5

Modules

Documentation for independent EPICS software modules: Record, Device and Driver Support, software frameworks, interfaces to other programming languages and applications.