EPICS pvAccess and Channel Access Documentation

Jul 21, 2020

General Documentation:

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.

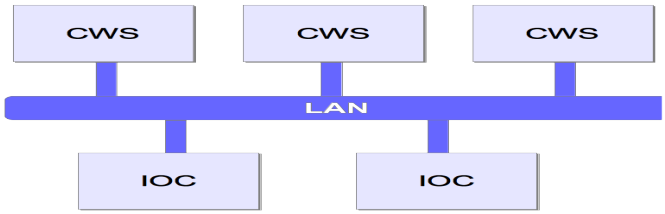

Figure 1. A simple EPICS control system structure.

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 to 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.

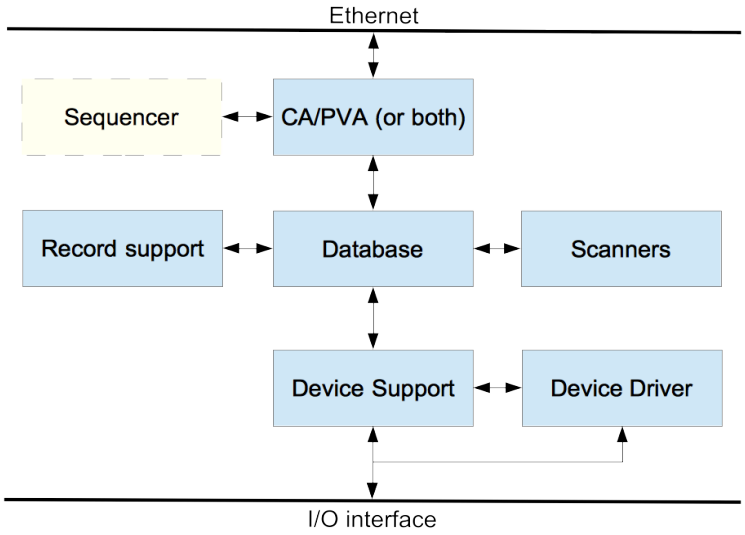

Figure 2. EPICS IOC components.

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).

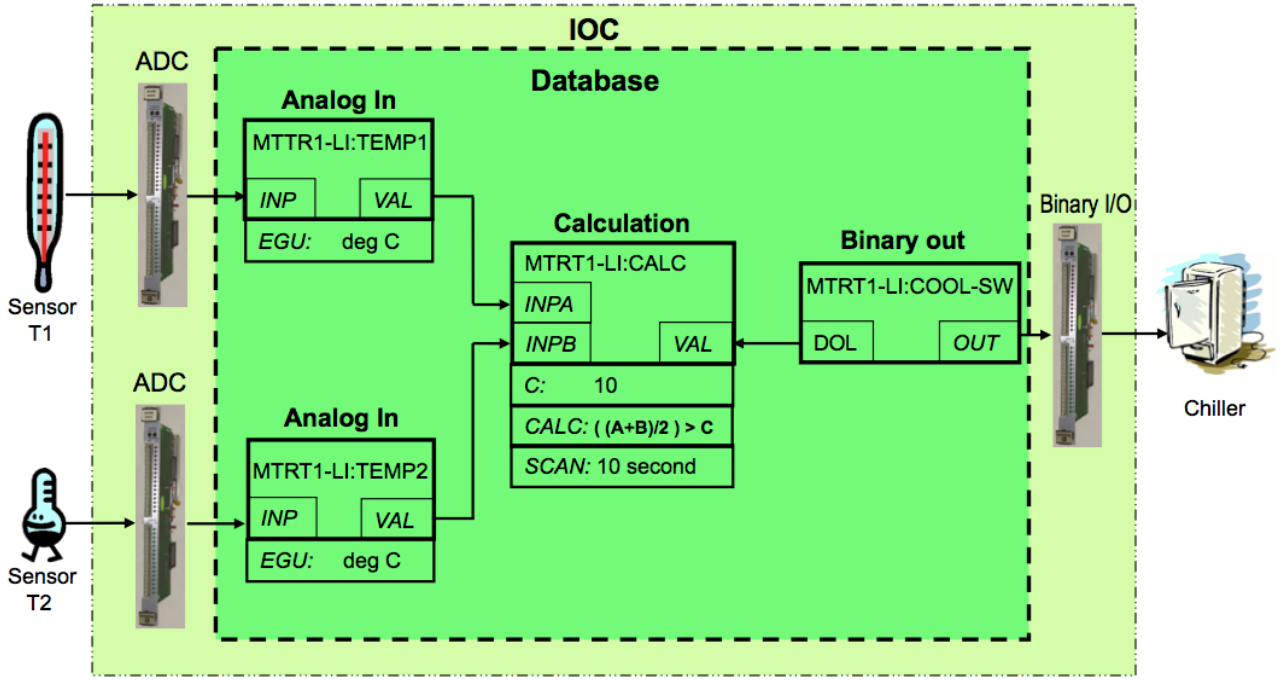

Figure 4. Example of record linking. From [2].

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.

1.4.2 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.

1.4.3 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.

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.
- 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.

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.

[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.

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

Channel Access Security Requirements

Ned D. Arnold

Table of Contents

2.1 Abstract

The Advanced Photon Source Control System is based on EPICS, the Experimental Physics and Industrial Control System co-developed by Los Alamos National Lab and Argonne National Lab. The basic architecture of EPICS allows for intelligent VME-based computers (referred to as Input/Output Controllers or IOCs) to be placed throughout the facility close to the APS equipment to be monitored and controlled. These IOCs are all interconnected to each other and to Unix-based workstations to allow physicists, engineers and technicians to monitor the operation of APS and to make changes to operating parameters from workstations in the control rooms or even their offices. The interconnection of IOCs and workstations is accomplished by widely accepted standards, namely ethernet and TCP/IP.

Extreme flexibility is provided by this "standards-based" interconnection mechanism, allowing users from all over the world to have access to APS data. However, significant security issues arise when it is realized that access to APS data also implies (currently) access to APS parameters that can effect the operation of the machine. This document discusses those security issues and then presents specific requirements for securing the APS Control System from unauthorized access.

A thorough Access Control System must accommodate the following observations:

- Certain individuals may be authorized to control some parameters but not others (e.g. a LINAC technician should not be permitted to adjust quadrupole currents in the Storage Ring)
- Many individuals may have permission to monitor (or read) machine parameters while only a few individuals will be authorized to modify (or write) them.
- Certain parameters must be altered only from specific locations (e.g. if someone is doing maintenance on the LINAC from an Operator Interface workstation in the klystron gallery, it would be quite inconvenient for someone in the Control Room to adjust the same equipment)
- The status of the Advanced Photon Source should be utilized in determining access authorization to a machine parameter. Certain modes of APS will require extreme control over what parameters may be adjusted (e.g. stored beam mode, storage ring orbit studies).

Therefore, the requirements for access control described in this document allow access to APS data and machine parameters based on four criteria: originator of the request (who); type of access, e.g. read or write (what); source (location) from which the request originated (where); machine status at the time of the request (when). Access rules predefined by qualified APS Operations Personnel will limit any access to the APS Control System to those specifically authorized. Machine parameters can be grouped together and different rules defined for each parameter group (allowing LINAC parameters to have different rules than Storage Ring parameters). Four access levels are used to grant increasingly more access to those that are appropriately authorized.

2.2 Introduction

The Experimental Physics and Industrial Control System (EPICS) allows for distributed control of very large facilities via software based on standard network protocols and interfaces. Input/Output Controllers (IOCs) attached to the network respond to commands and provide monitor data to "clients" that are also on the network. This distributed architecture has numerous advantages compared to a centralized topology, including modularity, expandability, low vulnerability to a single failure, etc. Security, however, must be specifically addressed to ensure that "unauthorized clients" do not unexpectedly alter critical setpoints or parameters during machine operation. This challenge is a bit more formidable in an open distributed system than in a closed centralized system.

Fortunately, there are only three conceivable paths by which one can alter the pre-programmed functionality within an IOC: by gaining access to the IOC operating system (vxWorks) shell using telnet via ethernet; by gaining access to the IOC operating system shell via the RS232 port on the CPU board; or by communicating with the EPICS software via itqs unique application layer network communication protocol called Channel Access.

This document specifies the requirements for ensuring that only "authorized individuals" can effect changes in the control system via the Channel Access path. The other vulnerabilities (access to the IOC operating system) are briefly discussed in this document to provide a unified presentation of the IOC security issue, but detailed requirements are presented in referenced documents.

2.3 IOC Access Control Requirements Overview

This section will introduce the requirements for the IOC Access Control implementation. The following discussion is not intended to present detailed functional requirements, but to present the general expectations of IOC Access Control.

2.3.1 Restrict Access to the IOC Operating System

Access to the IOC operating system must be restricted to authorized individuals who are trained in vxWorks and have a legitimate need to be working on the system at that level. Authorized individuals would include the application engineers (and trained technicians) responsible for that system or system software engineers that may be trouble shooting a software problem.

In addition, some mechanism must be provided to give authorized individuals access to the appropriate IOC independent of location. It is unacceptable to require the individual to be next to the IOC that requires attention.

If access to the IOC operating systems is securely limited to trained, cognizant, and authorized individuals, there is no need to make this access dependent on machine operating status. This authority level is comparable to a "super-user", and such classification requires responsible use of the authority allowed.

Preventing access to the IOC operating system is a system design challenge. Detailed requirements of the security requirements are <will be> discussed in the Functional Requirement "Access to The IOC Operating System".

2.3.2 Prevent IOC Access from Outside the APS Control System Subnet

Since the Channel Access protocol is built upon the popular TCP/IP suite of protocols, the possibility exists for Channel Access clients to reside anywhere in the world. This, of course, must be precluded.

Design of the control system communication network must restrict outside clients from direct access to the IOCqs. Several approaches are available and will be further discussed in other design documents.

The design should not absolutely preclude access to control system data from outside the control system "subnet". It should limit the who, what and how this data is obtained from the control system. For example, an authorized individual could telnet to the host computer on the control system subnet, start an authorized Channel Access client to collect data, and have that data returned to him.

This security issue is a combination of system design (network layout) and software utilities. Further details of the specific requirements are <will be> provided in the Functional Requirement "Channel Access Gateway to the Outside World".

2.3.3 Restrict Channel Access Requests to "Authorized" Clients

Channel access must restrict access to the IOC database parameters from all channel access clients except those specifically authorized to monitor or control that parameter. Authorized monitoring of database parameters should be quite lenient, but should be restricted when IOC performance is threatened. Authorized control or modification of a particular parameter is dependent on several factors.

A thorough IOC access control system would determine authorization to a particular database parameter based on four criteria: originator of the request (who); type of access, e.g. read or write (what); source (location) from which the request originated (where); machine status at the time of the request (when). The IOC Access Control implementation must balance the desired flexibility obtainable from these four parameters versus complexity of implementation and operation of the system.

- WHO: The first consideration for permitting control of database parameters is who is making the request. Clearly not all APS employees are qualified to adjust APS equipment via the control system. In addition, certain employees may be authorized to control some parameters but not others (e.g. a LINAC technician should not be permitted to adjust quadrupole currents in the Storage Ring). A mechanism must be provided to group authorized employees into authorization levels which Channel Access can use to allow or disallow a modification request. This also implies that the Channel Access client is able to determine who is initiating the requests.
- WHAT: Another consideration in allowing access to IOC database parameters is the type of access requested, e.g. read or write. In a typical control system environment, many individuals are likely to have permission to monitor (or read) database parameters while only a few individuals will be authorized to modify (or write) them. This flexibility must be provided in the Channel Access Security implementation.
- WHERE: Another consideration for permitting control of database parameters is where the request is from. An earlier requirement was to restrict "authorized clients" to those that are directly connected to the Control System Subnet, but additional flexibility in this respect is extremely advantageous. For example, if someone is doing maintenance on the LINAC from an Operator Interface workstation in the klystron gallery, it would be quite inconvenient for someone in the Control Room to adjust the same equipment. The location from where the request originates can generally be based on the IP number of the computer on which the Channel Access client is running, but the issue of "portable consoles" must be addressed as well.
- WHEN: Consideration of the status of the Advanced Photon Source must also be included in determining authorization into the IOC database. Certain modes of APS will require extreme control over what parameters may be adjusted (e.g. stored beam mode, storage ring orbit studies).

The consideration of the above criteria to determine access to the IOC database must be dynamically alterable by some appropriate administrative procedure. Ultimately, APS Operations will have overall control of the who, what, where, and when of IOC Access.

2.3.4 Discourage a Sophisticated Saboteur

The above requirements, if properly implemented, will discourage any inadvertent and/or direct attempt to interfere with normal APS operations. However, it is likely that the implementation will rely on existing security features in commercial hardware and software. It is beyond the scope of this effort to absolutely guarantee that no one can penetrate the access control scheme implemented to fulfill the above goals. There are no personnel safety issues involved, so the monumental task of implementing a system that can be guaranteed against even sophisticated saboteurs is not required for this application.

2.4 Functional Requirements

This section presents detailed requirements for Channel Access Security. Any discussions that imply a specific implementation are only suggestions used to clarify the requirement and are not binding on the implementer as long as the requirement is met. Refer to Figure 2 for an illustration of the requirements being discussed.

2.4.1 Enforcement of Channel Access Security

All requests between a "channel access client" and a "channel access server" must be dependent on pre-defined security restrictions described in the following paragraphs. This includes workstation-to-IOC communication as well as IOC-

to-IOC communication (that uses Channel Access). Process Variable "links" within an IOC that do not use Channel Access are not subject to these pre-defined access rules (e.g. dbget, dbput, etc).

2.4.2 Database Field Access Level

Each field of a record type will have an "access level" defined to it at the time that the record type is defined (in xxxxRecord.ascii). "Access level" is an entry from 1 to 4 representing different restraints that must be satisfied prior to allowing access to that field (i.e. each level can be assigned different \qaccess rulesq for granting permission to read or write from/to that particular field). Typically, higher access levels are more restrictive than lower access levels, but this is more of a convention than restraint, as will be seen later.

2.4.3 Process Variable Groups

Process variables (unique instances of any record type) will be grouped into PV Groups where each process variable in that group requires identical rules for each of the four access levels. There is no constraint on the number of PV Groups nor the number of process variables within a group. Any process variable can only be a member of one group.

2.4.4 PV Group Access Rules

Each PV Group will have a set of rules for each access level. The rules will define the prerequisite conditions (who, what, when, from where) that must be fulfilled prior to access being granted. The rules will be entered in the form of logical expressions that must evaluate to be true in order for the requested access to be granted. The right hand of the expression may contain logical operands, User Access Group names (UAGs), Location Access Group names (LAGqs), or Process Variables (PVqs). Examples are provided below:

```
Level 1 :  READ = *  /* all allowed to read fields with this access level */
WRITE = *  /* all allowed to write fields with this access level */
Level 2 :  READ = *
WRITE = UAG[linac]   /* linac group allowed at any time */
Level 3 :  READ = (PV[LI:IOCLTSC:caConnectionsSR] < 100)
WRITE = NONE  /* example for a video image */
Level 4 :  READ = *
WRITE = (UAG[linac] && (PV[LI:OP:stateCC] !=RUNNING) &&
             LAG[ICR])
A complete list of possible operands and operations follows:
OPERANDS :
UAG[example_1] : A predefined User Access Group named example_1. Refer to Section 4.5␣
↪.
LAG[example_2] : A predefined Location Access Group named example_2. Refer to Section␣
↪4.6 .
PV[example_3] : A process variable named example_3. Refer to Section 4.7 .
* : Wild card or don\qt care condition. Access always allowed.

OPERATIONS :
The following standard C operators must be supported:

||, &&, !=, <, >, >=, <=, == , !
```

2.4.5 User Access Groups [UAG]

Groups of individual users can be defined and then referred to by a UAG name. For example, all authorized linac operators could be defined in a group and then referred to by UAG[linac]. There is no constraint on the number of

User Access Groups nor the number of users within a group. An individual can be included in multiple UAGs. To indicate a particular user (instead of a group), that userqs name can be used instead of the UAG name (e.g. UAG[mrk] refers to an individual whoqs user name is mrk). For interactive channel access clients, provisions must be made to alter the current user (e.g. su nda) without requiring the client program to restart.

2.4.6 Location Access Groups [LAG]

Location Access Groups define particular workstations (using the name of the workstation) which are allowed access, based on the access rules. Groups of workstations can be defined and then referred to by a LAG name (e.g. UAG[InjectionControlRoom] or UAG[ICR]) . There is no constraint on the number of Location Access Groups nor the number of workstations within a group. If a particular workstation is not included in any LAG, that workstation can only access database fields that have no LAG entry in its PV Group Access Rule.

2.4.7 Process Variable [PV]

Process variables can be included in the PV Group Access Rules to implement access that is dependent on \qreal-timeq status of the machine. Should a change in a process variable occur such that access to a particular database field is inhibited, this change must must take effect within five seconds of the process variable changing to the new value. It is unacceptable to evaluate rules using process variables only at connection time.

2.4.8 Configuration Changes

Configuration changes in the Channel Access Security System will only be done by authorized "Operations" personnel. A mechanism for altering the rules, defining new Location Access Groups or User Access Groups, and forcing these changes to become immediately effective must be provided.

Analog Input Record (ai)

This record type is normally used to obtain an analog value from a hardware input and convert it to engineering units. The record supports linear and break-point conversion to engineering units, smoothing, alarm limits, alarm filtering, and graphics and control limits.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Input Specification

These fields control where the record will read data from when it is processed:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No

The DTYP field selects which device support layer should be responsible for providing input data to the record. The ai device support layers provided by EPICS Base are documented in the Device Support section. External support modules may provide additional device support for this record type. If not set explicitly, the DTYP value defaults to the first device support that is loaded for the record type, which will usually be the `Soft Channel` support that comes with Base.

The INP link field contains a database or channel access link or provides hardware address information that the device support uses to determine where the input data should come from. The format for the INP field value depends on the device support layer that is selected by the DTYP field. See Address Specification for a description of the various hardware address formats supported.

Units Conversion

These fields control if and how the raw input value gets converted into engineering units:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
RVAL	Current Raw Value	LONG	No		Yes	Yes	Yes
ROFF	Raw Offset	ULONG	No		Yes	Yes	Yes
ASLO	Adjustment Slope	DOUBLE	Yes	1	Yes	Yes	Yes
AOFF	Adjustment Offset	DOUBLE	Yes		Yes	Yes	Yes
LINR	Linearization	MENU (menuConvert)	Yes		Yes	Yes	Yes
ESLO	Raw to EGU Slope	DOUBLE	Yes	1	Yes	Yes	Yes
EOFF	Raw to EGU Offset	DOUBLE	Yes		Yes	Yes	Yes
EGUL	Engineer Units Low	DOUBLE	Yes		Yes	Yes	Yes
EGUF	Engineer Units Full	DOUBLE	Yes		Yes	Yes	Yes

These fields are not used if the device support layer reads its value in engineering units and puts it directly into the VAL field. This applies to Soft Channel and Async Soft Channel device support, and is also fairly common for GPIB and similar high-level device interfaces.

If the device support sets the RVAL field, the LINR field controls how this gets converted into engineering units and placed in the VAL field as follows:

1. RVAL is converted to a double and ROFF is added to it.
2. If ASLO is non-zero the value is multiplied by ASLO.
3. AOFF is added.
4. If LINR is `NO CONVERSION` the units conversion is finished after the above steps.
5. If LINR is `LINEAR` or `SLOPE`, the value from step 3 above is multiplied by ESLO and EOFF is added to complete the units conversion process.
6. Any other value for LINR selects a particular breakpoint table to be used on the value from step 3 above.

The distinction between the `LINEAR` and `SLOPE` settings for the LINR field are in how the conversion parameters are calculated:

- With `LINEAR` conversion the user must set EGUL and EGUF to the lowest and highest possible engineering units values respectively that can be converted by the hardware. The device support knows the range of the raw data and calculates ESLO and EOFF from them.
- `SLOPE` conversion requires the user to calculate the appropriate scaling and offset factors and put them directly in ESLO and EOFF.

Smoothing Filter

This filter is usually only used if the device support sets the RVAL field and the Units Conversion process is used. Device support that directly sets the VAL field may implement the filter if desired.

The filter is controlled with a single parameter field:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SMOO	Smoothing	DOUBLE	Yes		Yes	Yes	No

The SMOO field should be set to a number between 0 and 1. If set to zero the filter is not used (no smoothing), while if set to one the result is infinite smoothing (the VAL field will never change). The calculation performed is:

where `New Data` was the result from the Units Conversion above. This implements a first-order infinite impulse response (IIR) digital filter with z-plane pole at SMOO. The equivalent continuous-time filter time constant τ is given by

where T is the time between record processing.

Undefined Check

If after applying the smoothing filter the VAL field contains a NaN (Not-a-Number) value, the UDF field is set to a non-zero value, indicating that the record value is undefined, which will trigger a `UDF_ALARM` with severity `INVALID_ALARM`.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They do not affect the functioning of the record at all.

- DESC is a string that is usually used to briefly describe the record.
- EGU is a string of up to 16 characters naming the engineering units that the VAL field represents.
- The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields.
- The PREC field determines the floating point precision (i.e. the number of digits to show after the decimal point) with which to display VAL and the other DOUBLE fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No

Alarm Limits

The user configures limit alarms by putting numerical values into the HIHI, HIGH, LOW and LOLO fields, and by setting the associated alarm severity in the corresponding HHSV, HSV, LSV and LLSV menu fields.

The HYST field controls hysteresis to prevent alarm chattering from an input signal that is close to one of the limits and suffers from significant readout noise.

The AFTC field sets the time constant on a low-pass filter that delays the reporting of limit alarms until the signal has been within the alarm range for that number of seconds (the default AFTC value of zero retains the previous behavior).

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU (menuAlarmSevr)	Yes		Yes	Yes	Yes
HSV	High Severity	MENU (menuAlarmSevr)	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU (menuAlarmSevr)	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU (menuAlarmSevr)	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No
AFTC	Alarm Filter Time Constant	DOUBLE	Yes		Yes	Yes	No
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The monitors are sent when the current value exceeds the last transmitted value by the appropriate deadband. If these fields are set to zero, a monitor will be triggered every time the value changes; if set to -1, a monitor will be sent every time the record is processed.

The ADEL field sets the deadband for archive monitors (`DBE_LOG` events), while the MDEL field controls value monitors (`DBE_VALUE` events).

The remaining fields are used by the record at run-time to implement the record monitoring functionality.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No
ORAW	Previous Raw Value	LONG	No		Yes	No	No

Simulation Mode

The record provides several fields to support simulation of absent hardware. If the SIML field is set it is used to read a value into the SIMM field, which controls whether simulation is used or not:

- SIMM must be zero (`NO`) for the record to request a value from the device support.

- If SIMM is `YES` and the SIOL link field is set, a simlated value in engineering units is read using the link into the SVAL field, from where it will subsequently be copied into the VAL field.
- If SIMM is `RAW` the SIOL link is still read into SVAL, but is then truncated and copied into the RVAL field. The *"Units Conversion"* process described above is then followed to transform the simulated raw value into engineering units.

The SIMS field can be set to give the record an alarm severity while it is in simulation mode.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU (menuSimm)	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	DOUBLE	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU (menuAlarmSevr)	Yes		Yes	Yes	No

Device Support Interface

The record requires device support to provide an entry table (dset) which defines the following members:

```
typedef struct {
    long number;
    long (*report)(int level);
    long (*init)(int after);
    long (*init_record)(aiRecord *prec);
    long (*get_ioint_info)(int cmd, aiRecord *prec, IOSCANPVT *piosl);
    long (*read_ai)(aiRecord *prec);
    long (*special_linconv)(aiRecord *prec, int after);
} aidset;
```

The module must set `number` to at least 6, and provide a pointer to its `read_ai()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide an `init_record()` routine to configure the record instance and connect it to the hardware or driver support layer, and if using the record's *"Units Conversion"* features they set `special_linconv()` as well.

The individual routines are described below.

Device Support Routines

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

```
long init_record(aiRecord *prec)
```

This optional routine is called by the record initialization code for each ai record instance that has its DTYP field set to use this device support. It is normally used to check that the INP address is the expected type and that it points to a valid device; to allocate any record-specific buffer space and other memory; and to connect any communication channels needed for the `read_ai()` routine to work properly.

If the record type's unit conversion features are used, the `init_record()` routine should calculate appropriate values for the ESLO and EOFF fields from the EGUL and EGUF field values. This calculation only has to be performed if the record's LINR field is set to `LINEAR`, but it is not necessary to check that condition first. This same calculation takes place in the `special_linconv()` routine, so the implementation can usually just call that routine to perform the task.

```
long get_ioint_info(int cmd, aiRecord *prec, IOSCANPVT *piosl)
```

This optional routine is called whenever the record's SCAN field is being changed to or from the value `I/O Intr` to find out which I/O Interrupt Scan list the record should be added to or deleted from. If this routine is not provided, it will not be possible to set the SCAN field to the value `I/O Intr` at all.

The `cmd` parameter is zero when the record is being added to the scan list, and one when it is being removed from the list. The routine must determine which interrupt source the record should be connected to, which it indicates by the scan list that it points the location at `*piosl` to before returning. It can prevent the SCAN field from being changed at all by returning a non-zero value to its caller.

In most cases the device support will create the I/O Interrupt Scan lists that it returns for itself, by calling `void scanIoInit(IOSCANPVT *piosl)` once for each separate interrupt source. That routine allocates memory and inializes the list, then passes back a pointer to the new list in the location at `*piosl`.

When the device support receives notification that the interrupt has occurred, it announces that to the IOC by calling `void scanIoRequest(IOSCANPVT iosl)` which will arrange for the appropriate records to be processed in a suitable thread. The `scanIoRequest()` routine is safe to call from an interrupt service routine on embedded architectures (vxWorks and RTEMS).

```
long read_ai(aiRecord *prec)
```

This essential routine is called when the record wants a new value from the addressed device. It is responsible for performing (or at least initiating) a read operation, and (eventually) returning its value to the record.

... PACT and asynchronous processing ...

... return value ...

```
long special_linconv(aiRecord *prec, int after)
```

This optional routine should be provided if the record type's unit conversion features are used by the device support's `read_ai()` routine returning a status value of zero. It is called by the record code whenever any of the the fields LINR, EGUL or EGUF are modified and LINR has the value `LINEAR`. The routine must calculate and set the fields EOFF and ESLO appropriately based on the new values of EGUL and EGUF.

These calculations can be expressed in terms of the minimum and maximum raw values that the `read_ai()` routine can put in the RVAL field. When RVAL is set to *RVAL_max* the VAL field will be set to EGUF, and when RVAL is set to *RVAL_min* the VAL field will become EGUL.

The formulae to use are:

Note that the record support sets EOFF to EGUL before calling this routine, which is a very common case (when *RVAL_min* is zero).

Extended Device Support

...