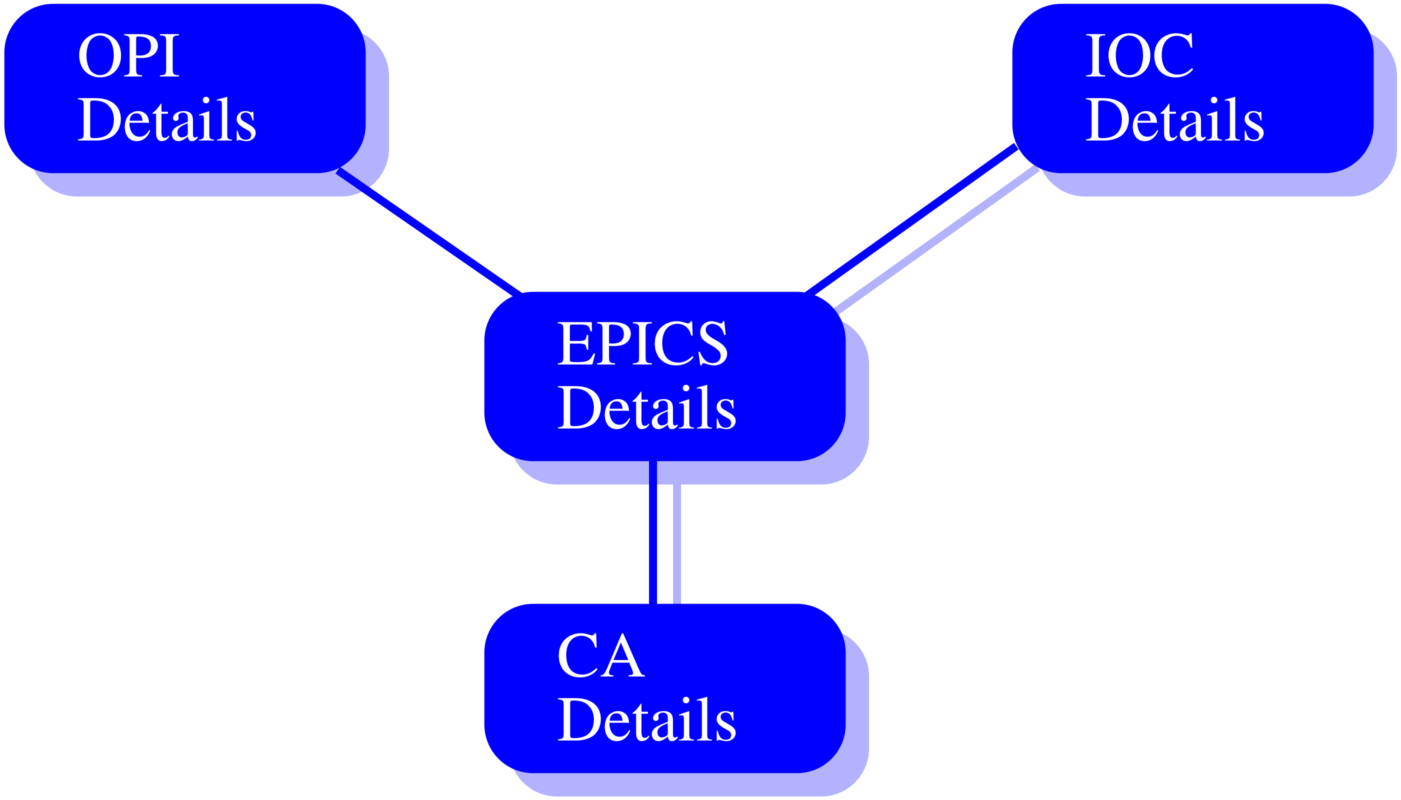
EPICS Detail

Intended for an audience of EPICS application developers.



EPICS Records.

• Ai, Ao	Analog In/Out. Read/write value from/to ADC/DAC.
• Bi, Bo	Binary In/Out. Read/write from/to digital hardware.
• Li, Lo	Long Word In/Out. Read/write from/to scalars/counters.
• Si, So	String In/Out. Store useful strings.
• Calc	Calculation Record. Arbitrary one-line processing.
• Compress	Compression/Circular Buffer/Min-Max-Avg container.
• Event	Generate Asynchronous Software Events
• Histogram	Acquire Histogram.
• Sub	Subroutine Record. Arbitrary 'C' language routine.
• Waveform	Waveform Record. Reads an array of data from Hardware.
• SubArray	Extract variable portion of Waveform array.
• D/Fanout	Fanout of Data/Fanout of processing links.
• Sel, Seq	Select among processing links; Sequence records with time delay.
• Scan, Wait	Multi-dimensional control of "detector" motion and operation.
• Timer, Motor	Control timing hardware; Drive stepper motor/encoder assembly.

...and many application-specific record types, *e.g.*, PID, HiV, pulse, vmeProbe.

Record Linking

Database records do not always exist in isolation; they may be linked to each other for processing flow and data exchange. This is a great strength of EPICS, but proper usage requries careful study by application developers.

Linkages between records are of the following types:

- INLINK An input link (gets the data).
- OUTLINK An output link (puts the data).
- FLNK A Forward link, propagates processing onwards.

INLINK and OUTLINK transfer data between records.

Links have two attributes, Process Passive and Maximize Severity.

Process Passive (PP) will cause the record at the end of the link to process if the SCAN attribute of the latter is Passive. The user may switch this off by casting the link as Non Process Passive (NPP)--the default.

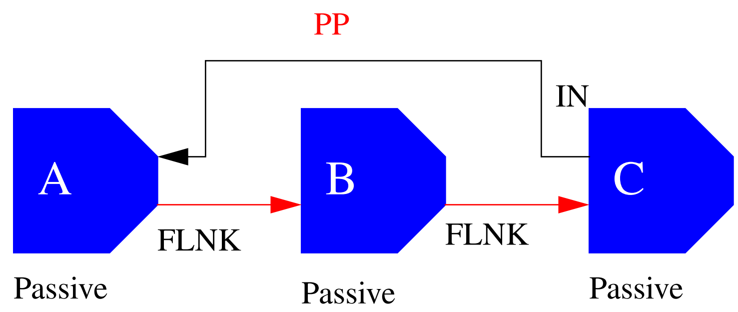
Maximize Severity (MS) will propagate alarms down a chain of records. This may be switched off by setting the link to Non Maximize Severity (NMS)--the default.

A Forward link does not move data, thus is only useful to cause processing of a another Passive record.

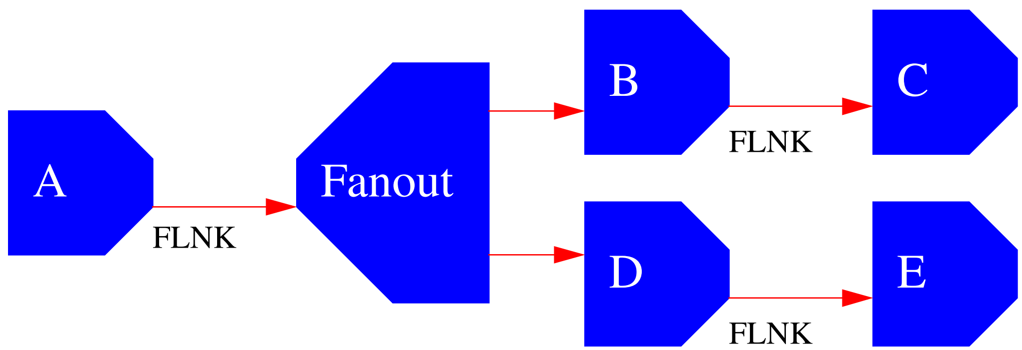
The PACT Field

Each record has a Processing Active Field (PACT). It is used to detect loops. Linked fields are treated as a “locked” (atomic) set. The processing order is deterministic.

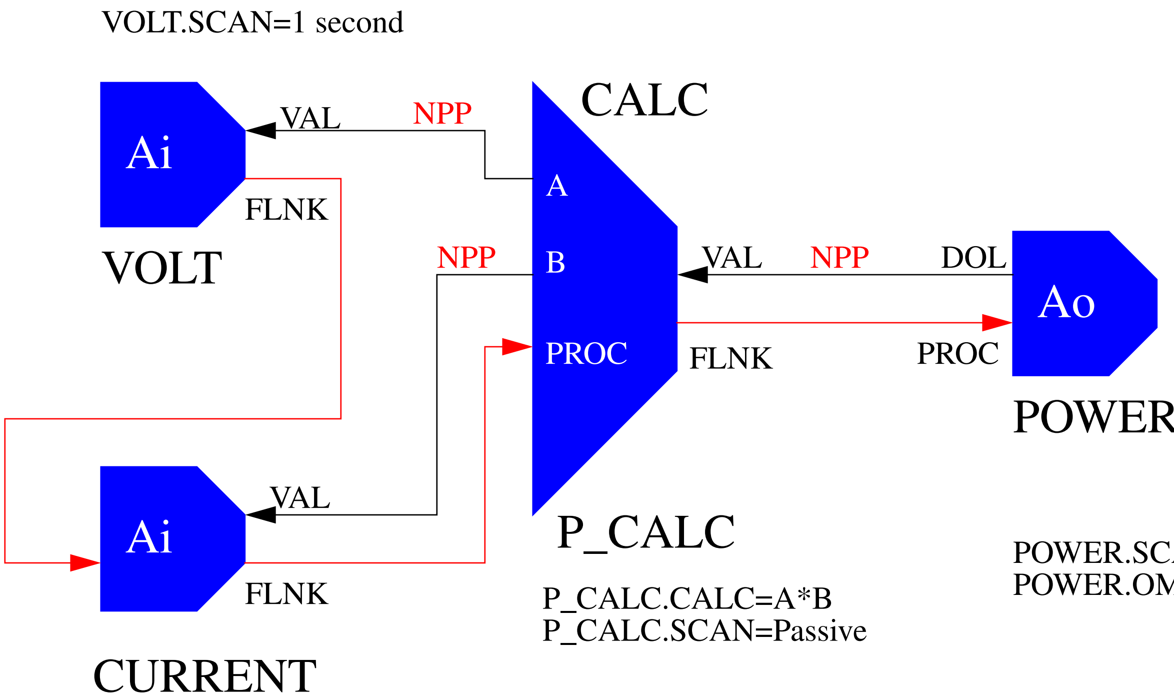
Loop Detection:



Processing Order:



Linking Example. 1A



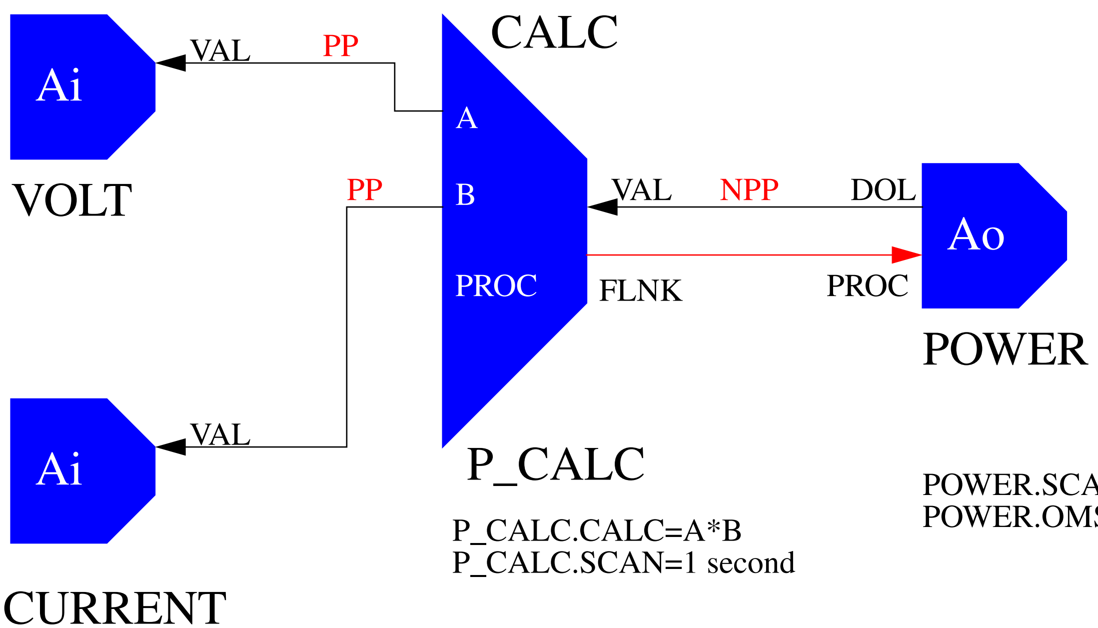
P_CALC.CALC=A*B
P_CALC.SCAN=Passive

POWER.SCAN=PASSIVE
POWER.OMSL=closed loop

CURRENT.SCAN=Passive

Linking Example. 1B

VOLT.SCAN=Passive

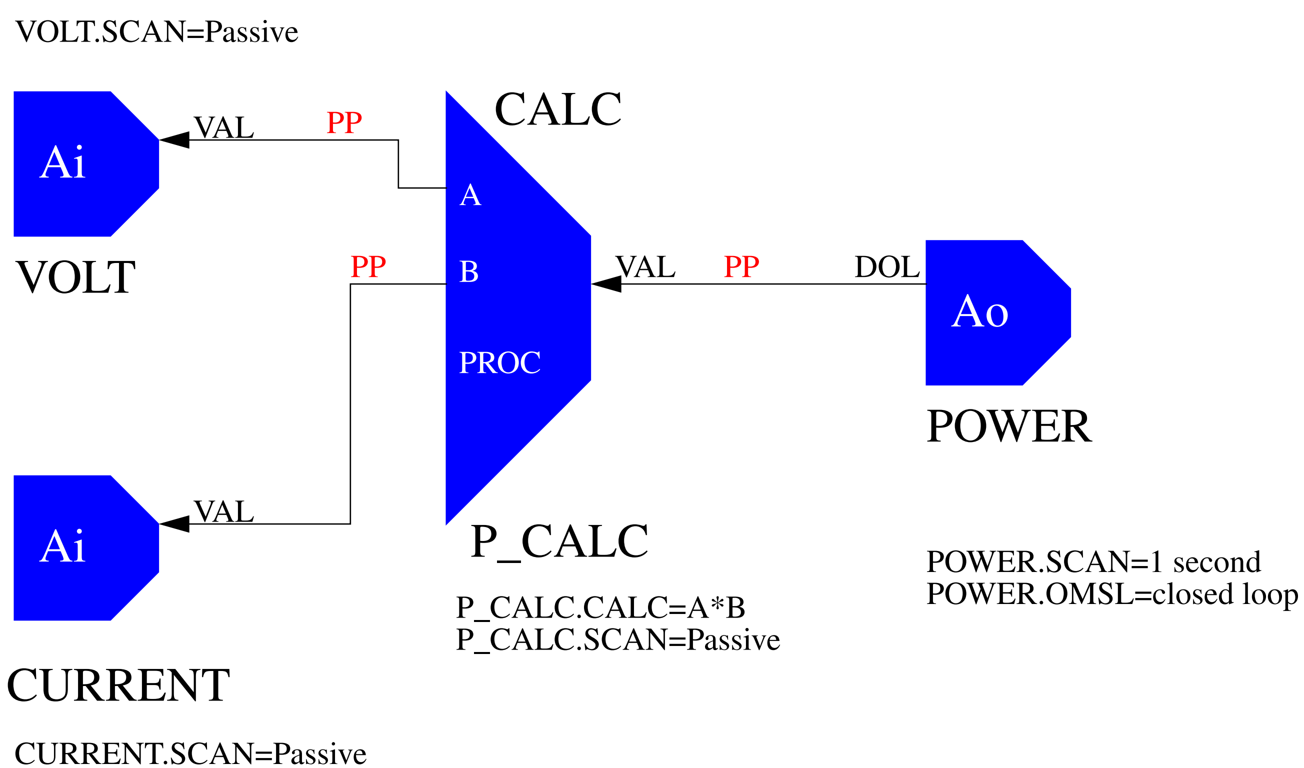


P_CALC.CALC=A*B
P_CALC.SCAN=1 second

POWER.SCAN=PASSIVE
POWER.OMSL=closed loop

CURRENT.SCAN=Passive

Linking Example. 1C



Link Resolution

When the database is downloaded onto the IOC, the link fields (INLINK, OUTLINK, FORWARD LINK) contain the Process Variable Link name (PV Link).

During IOC initialization, all records resident on the IOC are reviewed and the PV Links are resolved into:

- DB LINK — The linked record (channel) is local (in the same IOC), and is accessed via dbAccess.
- CA LINK — The linked record (channel) is remote (on another IOC), and is accessed via Channel Access.

An application's database does not necessarily have to be on the same IOC. However, if the developer decides to distribute the database over several IOCs, the relatively slow speed and non-deterministic nature of CA suggests that the number of links between database segments be minimized (loosely coupled).

Linked records on the same IOC comprise a Lock Set; all the records in a given Lock Set are processed atomically.

New options: CA forces a local link to use Channel Access semantics, *i.e.*, not be part of the Lock Set. Options CP and CPP allow processing of the INLINKing record when monitors occur.

Record Processing

Records are processed by the scan tasks. Scanning is performed under control of the record's SCAN field:

- Period — Scan periods of 10 second, 5 second, 2 second, 1 second, 0.5 second, 0.2 second or 0.1 second are standard*.
- I/O Event — Event raised by Interrupt Service routine†.
- Event — Software event (1-255), raised by task, sequencer, subroutine record, *etc*.
- Passive — Processed if Channel Access put to VAL (or other) field, or linked via Process Passive to a processed record (INP/OUT), or if in a forward linked chain (FLNK).

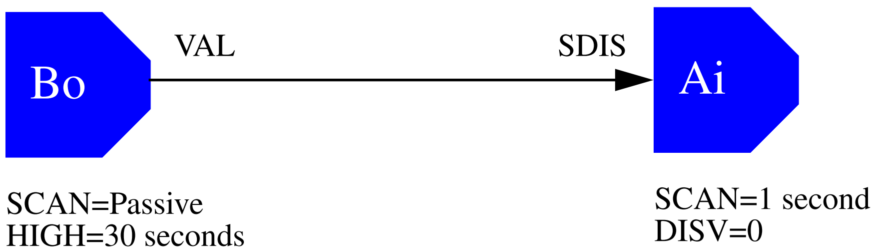
Any record will be unconditionally processed by writing to its PROC field.

When a record is first processed, the Undefined field UDF is set to zero. Whenever it is processed, its TIME field is updated. record can also be processed, once, at initialization using the PINI field.

*Rates can be changed and more can be added.
†When used with supporting DevSup/DrvSup.

Scan Disable

Some applications require that a record be disabled at certain times. For example, consider an input record which is normally non-active, but for certain periods of time it is required to run periodically.



Record fields used by scan disable:

- SDIS Scan Disable Input link. Where the disable value is obtained.
- DISA The scan disable value read from SDIS.
- DISV If DISA=DISV, then the record is disabled.
- DISS The alarm severity to be raised when the record is disabled.

When a disabled record is processed, it returns immediately and performs no actions.

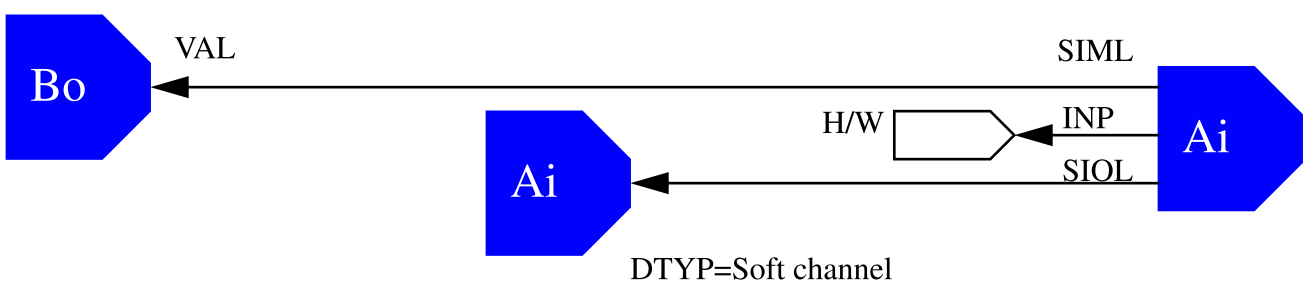
Simulation Mode

When creating an application, the developer may wish for the database to be tested before the hardware interface has been completed.

I/O records have a simulation mode, enabling the record to read/write its value from/to another record instead of hardware.

The simulation mode fields are:

- SIOL — Simulation value location, the location used to read/write in place of the hardware INP, OUT address.
- SIML — Simulation mode location, where the mode is obtained.
- SIMM — The simulation mode value read from SIML.
- SIMS — The alarm severity to be raised when the record is in the simulation mode (may be disabled using NO ALARM).



Record Support

Record support differs for different record types, but they all perform the following general steps:

- Check that record is not already being processed (PACT≠1).
- Check that record is not disabled.
- Set PACT=1 while processing.
- Perform I/O using Device Support and optionally Driver Support.
- Check for record-specific alarm conditions.
- Raise database monitors.
- Request processing of forward links.
- Unset PACT field and exit.

Additional “puts” to a busy record will cause the record to reprocess *once* more; valuues are cached, *not* queued.

Asynchronous DevSup

Record Support processing is complicated by the fact that some devices are intrinsically asynchronous.

It is never permissible for RecSup to wait for a slow device to complete.

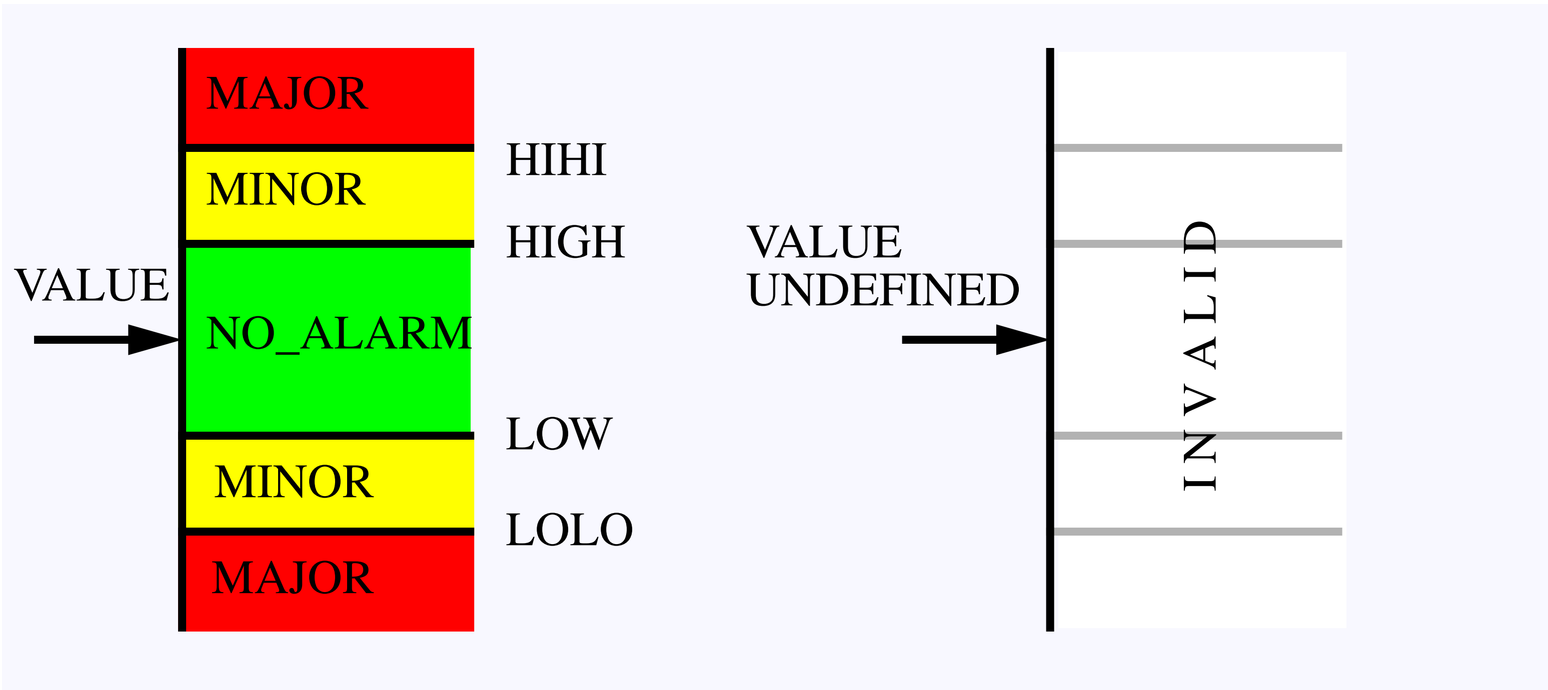
To allow DevSup to inform the RecSup that the device is in fact asynchronous, the PACT field is not set until after DevSup has returned.

```
static long Process() {
    old_pact = PACT;
    DevSup();
    if (!old_pact && PACT) return(0);
    PACT=TRUE;  /* second pass */
    alarm();
    monitor();
    recGblFwdLink();
    PACT=FALSE;
    return(status);
}
```

Alarm Processing: Status

Most alarms are raised if the record's value has exceeded a pre-defined boundary, or if the value is undefined (*e.g.,* device failed or missing).

There are four pre-set boundaries. Each can raise an alarm of a predefined **severity**. Allowed severities are (in decreasing order): INVALID, MAJOR, MINOR, NO_ALARM.



Alarm Processing: Status

The alarm **status** encodes the reason for the alarm, such as:

- READ: Failure to initialize or read input device
- LOW/HIGH: Value is too low/high
- WRITE: Failure to initialize or write output device
- STATE: Device state is defined to require a severity (*e.g.*, “Trip” is MAJOR)
- LINK: (Channel Access) Link not valid (*e.g.*, disconnected, non-existent)
- ACCESS: Channel Access permission denied
- UDF: Record undefined (never processed)
- DISABLE: Record is disabled
- SIMM: Record is in simulation mode
- COS: Change of hardware state occurred

The quiescent status/severity is NO_ALARM/NO_ALARM.

RecSup Monitors

The RecSup procedure monitor() will generally check if the value has changed by a sufficient amount to warrant the sending of the new value. Separate dead-bands are provided for displays, for archiving/logging, and for alarms. Fields concerned with this decision are:

- VAL New value
- MLST Value when monitors were last raised
- MDEL Monitor dead-band for displays.
- ADEL Monitor dead-band for archiving
- HYST Monitor dead-band for alarms

A Channel Access client can arm the monitor for any combination of the above.

For example, DM specifies MDEL AR specifies ADEL; ALH specifies HYST.

Record Support Declaration

The records which are available to the EPICS system are defined in the ascii files xxxRecord.dbd. Each lists all available fields and their attributes. A common file is “included” for shared fields; then record-specific fields follow.

A typical file looks like:

```
recordtype(ai) {
include "dbCommon.dbd"
field(VAL,...)
...
}
```

All such files are themselves included into a master file which is loaded into the IOC:

```
include "aiRecord.dbd"
...
```

Other objects in the database, such as menus and breakpoint tables, are defined in a similar way.

Device Support

I/O Records require a device support. The fields concerned with DevSup are;

- DTYP Device Type.
- INP/OUT Hardware address information for device type.

The Device Support code is completely written by the application developer. Devices available to the system are defined in the ascii files devXXX.dbd. Each is a list of the devices available for each record, the basic I/O type, the name of the Device Support Routine, and the corresponding DTYP name:

```
device(ai,VME_IO,devXXX,"XXX")
```

Device definitions may be collected into convenient bundles using “include” entries.

The format required to specify the Hardware address is device-specific, however, many devices use common formats. For example, `VME_IO` requires `card` and `signal`.

Driver Support

Use of DrvSup is entirely at the discretion of the application developer. If the low level device interface is relatively simple, then devSup may handle the hardware I/O.

The application developer defines a set of routines available to DevSup. They usually called directly from DevSup.

Driver Support is defined in the ascii files drvXXX.dbd. Example entries are:

```
driver(drvAb)
driver(drvCompuSm)
```

Driver and device definitions may be collected into related bundles and using “include” entries.

All record, device, and driver “.dbd” files are typically concatenated into a

master file specific to each IOC. It is this file which is loaded via dbLoadDatabase() when the IOC is initialized.

The EPICS collaboration maintains a body of shared Record, Device and Driver Support, and a WWW page refers to many locally maintained varieties.

Record & Device Initialization

Initialization is performed at two levels, general and record specific. The general level will initialize a device *type* or a record *type*. The record specific level will initialize each *instance* of a record. The ordered steps are:

1) Initialize Record Support. Call init() for each record type defined.

2) Initialize Device Support. Call init(after=0) as first pass for each occurrence of device support in the devSup.ascii file (*e.g.* for Ai, Ao, Bi, etc.).

3) Loop through database, initializing rset, dset. Call init_record(pass=0) (normally performs no operation).

4) Loop through database, resolve PV links to DB or CA links.

5) Loop though database, calling RecSup init_record(pass=1).

6) RecSup init_record calls DevSup init_record(recptr).

7) DevSup init called again with after=1.

IOC Boot

When each IOC is powered up, it should execute a start-up script (using the VxWorks shell) which will download and run the EPICS system. The start-up will incorporate the following steps;

- Download external object code, *e.g.* CAMAC library.
- Download EPICS Code; iocCore, drvLib, recLib, devLib, seq.
- Optionally download special initialization code initLib.
- Download subroutine records' object code.
- Download the database using dbLoadDatabase and instances of dbLoadRecords and dbLoadTemplate.
- Download sequencer object code.
- Initialize EPICS using iocInit "resource_def_file".
- Start sequencers and any other VxWorks tasks.

The resource definition file provides the equivalent to "environment variables" for VxWorks. These control parameters such as Time Zone, Error Logging Host, Network List, *etc*.

Channel Access Details

Each IOC has a CA Server running which is aware of all the records local to it.

A remote CA Client must first search for the channel(s) it is interested in by issuing a ca_search() call. This sends a UDP broadcast message over the LAN. Each IOC server reads the message and checks the channel name(s) against its database. If a match is found, the server acknowledges and checks to see if communications with the OPI has been previously started. If not, then a pair of tasks (ca_get, ca_put) are started which establish a pair of TCP/IP sockets and buffers connected to the client.

If a channel is duplicated over more than one IOC, then the first to acknowledge gets the connection. The CA Client is informed of the name duplication when the other acknowledgments are received.

I/O proceeds with ca_put() and ca_get() calls.

A client can register itself with an IOC's monitor system using the ca_add_event() call. This adds the client to the list of interested parties if a channel's value or alarm status changes. A client can also request a callback when a ca_put() cascade completes.

IOCs send beacons using UDP broadcast; clients use changes in beacon frequency to dynamically disconnect/reconnect from/to channels.

Access to every field of every record can be controlled. Three levels are access are available: NONE, READ-ONLY, READ/WRITE.

Channel Access Security

Records are gathered into Access Security Groups. For each group, an equation controls when each access level is asserted, based on:

- Client UID's
- Client IP address
- Values from *other* Channels

Every get/put access is checked.

The access equation may also be dynamically "put".

Application Development

EPICS sites have implemented a variety of application directory *frameworks*, covering very simple cases up through very complex situations, especially where control over record and device/driver support is desired on a per-IOC basis.

The application developer chooses which EPICS release and/or which VxWorks kernel to use. Soft links and Unix enviroment variables are used to implement this.

In the LBNL version ("SAFE"), for example, the structure is built using the `app-MakeDir` and `iocMakeDir` commands, which create and populate directories which hold:

- custom configured VxWorks kernels;
- sequencer and subroutine source files;
- start-up scripts and resource definition files;
- database files;
- display manager, alarm handler, archiver, and backup/restore configuration files;
- special "makefiles" to combine record, device, and driver data structures from the generic EPICS tree with local additions/replacements.

APS supports a comprehensive scripting environment called `makeBaseApp`.

Database Generation

Several different tools available for database creation.

The currently available tools are:

- DCT — Simple, single-record fill-in-the-blanks utility (TCL/Tk-based).
- GDCT — Graphical Database Configuration Tool (Interviews-based); record links are “wired”.
- Capfast — LANL/CEBAF modification of a commercially available Electronic-Schematic Capture Editor featuring hierarchy and templates.
- Perl/Awk — Scripts; may be used in conjunction with other packages.
- Vi... — Direct generation of database using your favorite editor...

The application developer should be aware of all available tools, and their strengths and weaknesses, before deciding which to use. Also, a combination of tools may best suit a particular application.

Database Generation

Each record in the database has an entry. Its fields are listed, line by line. If a field takes the default value, then it does not need to be included.

```
record(ao, "$(I):Control_A")
{
    field(DESC, "Supply Analog Control")
    field(SCAN, ".1 second")
    field(DTYP, "VMIVME-4100")
    field(OUT, "#C0 S3 @")
}
```

Just as the concatenated Record Support, Device Support and Driver Support ".dbd" file is used by the IOC to supply load-time definitions and defaults, it is also used by some configuration tools (DCT, GDCT) to prompt for and constrain input.

Database Loading

The ascii database is directly loaded into the IOC. After loading the defaults file as

```
dbLoadDatabase(master.dbd)
```

there are three options:

- Explicit — The non-default fields are explicitly specified for every record instance.

```
dbloadRecords(instances.db)
...
```

- Mixed — Some fields are specified, but others which have been left parametric have simple text substitution performed.

```
dbloadRecords(instances.db, "<par>=<val>, ...")
...
```

- Template — A file containing a hierarchical set of substitution/expansion rules and parameters lists is referenced, and only pure templates are supplied.

```
dbloadTemplate(template.tmpl)
...
```

Note that the records in any IOC may be loaded in logical “chunks” and need not be combined into a single file.

Sequencer Code

Sequencer code is compiled into C code using the State Notation Compiler (SNC). This allows the user to easily write state driven code which can attach to records for the purposes of reading, writing or monitoring their values.

```
program test
int voltage;
assign voltage to “volt”;
monitor voltage;
state run {
     when (voltage > 5){
          send_warning();
     } state shut_down
}
state shut_down {
     when(1) {
          perform_shutdown();
...
```

EDD/DM

EDD and DM are the original EPICS Display Editor and Manager. They were produced by Los Alamos and run on any X-windows workstation.

Features.

User can quickly and easily put together a display system; good tutorial.

Rapid switching launch of DM from EDD makes modifying the screen easy.

Product is directly supported by EPICS and requires no licenses.

Good interface with Channel Access, very conservative of resources.

Loads screens very fast (<0.5 sec); updates extremely fast (>5000/sec).

Widgets are simple but align and stack very well; cosmetic “bezel” available.

Can edit color-maps directly; can be explicitly shared.

Color “rules” available.

Support for drag-and-drop.

Can call up arbitrary shell commands.

Background pictures available.

Recursive macro nesting minimizes effort and maximizes uniformity.

Tcl/Tk

Tcl is an interpreted scripting system which runs on UNIX workstations. Tk provides a windowing toolkit underneath Tcl code. Higher order toolkits are also available, *e.g.*, BLT provides bar chart and graphs.

Advantages.

Complete set of widgets available.

Customisation of new widget types is easy.

Full window management available, e.g. iconisation, deletion of child widgets.

Attractive displays easy to construct.

Disadvantages.

Interpreted code, runs slowly on low powered systems.

Channel Access interface is built by CEBAF, subject to modification. Is not truly event driven, uses a polling algorithm.

Text-based (has syntax and semantics). (MEDM and EDD/DM are pure point-and-click).

Alarm Handler

The Alarm Handler(ALH) presents a responsive, hierarchical view of the alarm status of a large number of channels and allow the operator to selectively "acknowledge" them.

Features:

- Supports browsable hierarchy.
- "Guidance" text available per item.
- "Command" available per item, *e.g.*, pop-up small display manager screen or put a value to a channel.
- Suppression of any partial hierarchy or item based on value of any channel.
- Logs alarms, acknowledgements, and configuration changesavailable in persistent, viewable windows.
- Audible signal available.
- High performance (>1000 updates/sec).
- Configurated with an ascii file.

Alarm Handler, cont'd.

```
GROUP NULL TOP
$COMMAND <command>
$GUIDANCE
.........guidance text............
$END
CHANNEL TOP channel_0
GROUP TOP MIDDLE
CHANNEL MIDDLE channel_1
```