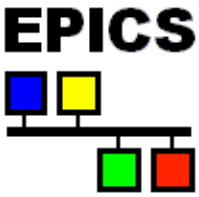


EPICS IOC Software Configuration Management

Releases 3.13.1 and 3.13.2

Marty Kraimer, Andrew Johnson, Janet Anderson, Ralph Lange

April 2000

EPICS

Table of Contents

Table of Contents

Table of Contents

1. Overview

Several EPICS Application Source/Release systems are available. Your site may have adapted one of them. Consult your EPICS system manager. This manual describes procedures that can be used for simple or complicated applications.

This document describes how to create and build IOC applications. This includes:

- IOC databases.
- vxWorks startup files.
- State Notation Programs.
- Record/Device/Driver support.
- Access security configuration files.
- Other code to be executed in an IOC.
- Special Host code.

Procedures are described for managing large applications. The principle features are:

- The IOC software is divided into `<top>` areas. Each `<top>` is maintained separately.
- Different `<top>` areas can be linked to different releases of software outside of that `<top>` such as EPICS base.
- Two basic classes of `<top>` exist:

  `<supporttop>`
  : A `<top>` the products of which are meant to be used by other `<top>` areas. Examples are EPICS base and various unbundled hardware support modules.

  `<ioctop>`
  : A `<top>` for a set of iocs which share the same applications.
- Where two or more `<top>` areas are linked, the linked versions must be coherent (i.e. all the same release of EPICS base and other software).
- All editable files are placed under CVS control.
- Each application developer makes changes in a private copy of one or more `<top>` areas.
- Operations maintains the official `<top>` areas from which the operational iocs are booted.
- `cvs`, `gnumake`, and `makeBaseApp` are the main tools used to create and build applications.
- Templates are provided for creating new application directories. EPICS base provides two sets of templates: simple and example. Additional templates can be created.

The structure of the EPICS CVS repository is described, along with the distributed maintenance approach which will apply to all hardware support modules in the future. The responsibilities of a module owner are also covered here.

User Prerequisites

This manual assumes that the reader:

- Understands the C language
- Knows how to use a text editor
- Has at least a superficial knowledge of the make utility

System Prerequisites

Before you can generate EPICS IOC applications your host and/or EPICS system manager must have done the following:

- Installed vxWorks and a board support package. Consult the vxWorks documentation for details.
- Installed EPICS base.

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.

1.1. Overview of Application Structure

A `<top>` has the following structure:

```
<top>/
    config/
    xxxApp/
        src/
        xxxSrc/
        ...
        Db/
        xxxDb/
        ...
    xxxApp/
    ...
    iocBoot/
        iocxxx/
        ...
    bin/
        <host_arch>
        <target_arch(s)>
    include/
    db/
    dbd/
    ...
```

Each `<top>` area is a separately managed set of application files. Separately managed means that each `<top>` can use it's own release of software obtained from outside the application, e.g. a release of EPICS base.

A `<top>` area may contain one or more xxxApp subdirectories and at most one iocBoot directory. The xxxApp areas are created by application developers as needed to manage the software in a logical structure. All software components needed by IOCs are built in these xxxApp directories. The iocBoot directory contains a subdirectory for each IOC that boots from this `<top>` area. An IOC can only use a single `<top>` area for booting, and only one IOC should boot from each iocxxx subdirectory under iocBoot. The

most important source file in a boot directory is the `st.cmd` file which is executed after vxWorks is started on an IOC (the vxWorks boot parameters should have the full path to this file as the startup script). The `st.cmd` file loads various files which were built in the xxxApp directories and installed in the `bin/<arch>` or other installation directories, then starts the IOC software.

Application developers decide what constitutes a `<top>`. For example, at APS, the Linac is completely contained in a single `<top>` area, while the RF is spread over three `<top>` areas: parrf, boosterrf, and srrf. No `<top>` area contains iocxxx directories from multiple subsystems. A decision on how to divide a control system between different `<top>` areas and xxxApp directories within each `<top>` may be obvious for purely technical reasons. Where this is not the case also consider the effects of future upgrades and maintenance, for both the IOC applications and their support software.

1.2. Use by Application Developers

Under xxxApp are source and database directories. A source directory contains source files for building executables and database description files. Makefiles specify what to build. These makefiles can specify components obtained from outside the `<top>` area, such as EPICS base. After modifing files in this directory the command:

```
gnumake
```

will rebuild components that depend on the modified files. (It will be seen below that the generated components are actually copied to an install directory)

The xxxApp database directories contain IOC database files and/or template and substitution files used to generate database files. If templates are used then whenever a template or substitution file is modified the `gnumake` will recreate the database file(s).

After changes are made in any xxxApp directory and the components rebuilt, the affected IOCs can be rebooted from the appropriate iocBoot subdirectory. If new components are added, it may be necessary to modify the `st.cmd` files.

`<top>/config`

This directory contains make rules, configuration files, and file `RELEASE`. `RELEASE` contains a list of the locations of all products needed by this `<top>` that are maintained elsewhere (normally these are `<supporttop>` areas, see next section).

Types of Top

There are two different flavours of `<top>` described in this document which are used for slightly different purposes:

`<supporttop>`
: A `<supporttop>` contains products meant for use by other `<top>` areas, incuding other `<supporttop>` and `<ioctop>` areas. If a `<supporttop>` contains an iocBoot directory its only purpose is for use in testing the `<supporttop>` software, not for

operational systems. All files meant for use by other <top>s are installed into standard subdirectories such as include, bin and dbd.

<ioctop>

An <ioctop> has an iocBoot directory with an iocxxx sub−directory for each ioc. This contains the st.cmd file for the ioc, and can also be used for other ioc−specific files needed by the application.

Templates

The makeBaseApp.pl utility creates new application areas. It does this by copying and transforming directory trees from a template area. EPICS base provides templates for a simple application and for an example application. Each site can, however, create their own set of templates.

Tools

The following tools are used:

- cvs
- gnumake − GNU make plus EPICS supplied configuration definitions, make rules, and script files.
- makeBaseApp − An epics/base supplied tool for creating <top> areas.

References

Version Management with CVS, Per Cederqvist et al.

GNU Make, Richard M. Stallman and Ronald McGrath

EPICS Application Developer's Guide, Marty Kraimer

2. Getting Started

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

Check Environment

Execute the command:

```
echo $HOST_ARCH
```

This should display your workstation architecture, for example `solaris`. If you get an "Undefined variable" error ask your EPICS system manager how to set this variable for your particular environment.

Create example Application

Execute the commands:

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

where:

<top>
: Any directory name you chose.

<base>
: Full path name to EPICS base.

<arch>
: Your host architecture (i.e. the output of the echo command above).

For example at ANL/APS the following commands create an application:

```
cd
mkdir myapp
cd myapp
/usr/local/epics/baseR3.13.1/bin/solaris/makeBaseApp.pl -t example first
/usr/local/epics/baseR3.13.1/bin/solaris/makeBaseApp.pl -i -t example target
```

Windows Users Note:

Perl scripts are invoked with the command `perl <scriptname>` on win95/NT. Perl script names are case sensitive. For example to create an application on WIN95/NT:

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

Inspect Files

Spend some time looking at the files that appear under `<top>`. Do this **BEFORE** building.

Build

In directory `<top>` execute the command:

```
gnumake
```

Inspect Files

Again look at all the files that appear under `<top>`.

Boot Parameters

The next step is to set the IOC boot parameters via the console serial port on your IOC. Life is much easier if you find out how to connect the serial port to a window on your workstation. See your EPICS system manager for details.

The vxWorks boot parameters look something like the following:

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

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

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

Boot

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

Test

See the description of the example given in the section Application Templates Supplied with base. Also try some of the vxWorks shell commands described in the "IOC Test Facilities" chapter of the Application Developer's Guide.

3. Managing a <top>

3.1. Definition of <top>

<top>:

An autonomously managed area which has a structure like that created by makeBaseApp.

Autonomously managed means that each <top> can follow its own release schedule.

The locations of software that a <top> uses from outside its own area are specified in its config/RELEASE file. For example:

```
#RELEASE Location of external products
SUPPORT=/usr/local/iocapps/R3.13.1/support
TEMPLATE_TOP=$(EPICS_BASE)/templates/makeBaseApp/top

SHARE=$(SUPPORT)/share/1-1
MPF=$(SUPPORT)/mpf/1-1-asd1
EPICS_BASE=$(SUPPORT)/base/3-13-1-asd2

IMAGEFLOW=/net/phoebus/usr3/imageflow2.4/imageflow
MSITOP=/usr/local/epics/extensions
```

In the above example:

- IMAGEFLOW and MSITOP are obtained from outside the IOC applications area.
- Release numbers such as 1-1-asd1 are explained later in this document.
- The order of entries in the file is important when searching include files. EPICS_BASE should usually be listed last to allow other support areas to override its entries.

For ioc applications there are two flavors of <top>:

<ioctop>

A <top> area in which a set of related iocs are managed and booted.

<supporttop>

A set of software meant to be used by one or more <ioctop>s, e.g. device and driver support for a particular hardware interface board, or a commonly used database template. See the Support Management: Overview section for details.

It is expected that a small set of application engineers will be responsible for a particular <ioctop>, and that the iocs in an <ioctop> are responsible for a limited set of functions. If this is done it is much easier to manage ioc applications.

3.2. Directory Structure

```
<top>/
    Makefile
    config/
*       CONFIG
*       CONFIG_APP
        Makefile
*       RELEASE
        RULES.Db
        RULES.Host
        RULES.Vx
        RULES.ioc
        RULES.iocBoot
        RULES_ARCHS
        RULES_DIRS
        RULES_TOP
        makeDbDepends.pl
        makeIocCdCommands.pl
        replaceVAR.pl
    xxxApp/ or xxxapp
        Makefile
        src/ or xxxSrc/ or xxxsrc/
            Makefile
*           Makefile.Host
*           Makefile.Vx
*           base.dbd
*           baseLIBOBJS
*           <app>Include.dbd
*           C and State Notation Language source
*           menu, recordtype, device and driver database definitions
        Db/ or db/ or xxxDb/ or xxxdb/
            Makefile
*           record instance files
*           record template and substitution files
        privately managed directories
    iocBoot/ or iocboot/
        Makefile
        Makefile.Host
        nfsCommands
        iocxxx/
*           Makefile
*           st.cmd
$           cdCommands
$   dbd/
        installed database description files
$   db/
        installed record instance files
$   include/
             installed include files
$   bin/
$       <host_arch>
            installed Host executables
$       <ioc_arch>
            installed IOC products
$   lib/
```

Files above marked "*" are user created and/or edited. Each of these file types is discussed in this section.

Entries marked "$" are directories or files created by gnumake. Since `gnumake clean uninstall` removes these files and directories, no permanent data should be stored here.

Directories

The user−modifiable directories are:

config
: Directory containing configuration files for gnumake.

xxxApp
xxxapp
: Directory containing source files and database files. An arbitrary number of `xxxApp` directories are allowed. Each must have `App` or `app` appended to the name because the top−level Makefile looks for this pattern.

  xxxsrc
  xxxSrc
  : Directory containing source files. An arbitrary number of source directories can appear under each xxxApp. The names must have the suffix `src` or `Src`. A source directory is where C code, sequence programs, scripts, etc. are created and built.

  xxxdb
  xxxDb
  : Directory containing record instance files. An arbitrary number of Db directories can exist under each xxxApp. The name must have the suffix `db` or `Db`. Each Db directory can contain record instance, template, and substitution files or the equivalent CapFast schematics.

iocBoot
iocboot
: Directory containing a subdirectory for each IOC.

  iocxxx
  : Directory from which IOC `iocxxx` is booted. Each must have `ioc` prepended to the name because `iocBoot/Makefile` looks for it.

The following directories are created by `gnumake` to hold built objects:

dbd
: Installed Database Definitions Directory.

include
: Include Directory. The directory in which include files generated from menu and record type definitions are installed.

bin

Binaries Directory. This directory contains a subdirectory for the host architecture and for each target architecture. These are the directories in which executables, binaries, etc. are installed.

lib

Library Directory. This directory contains a subdirectory for the host architecture and for each target architectiure. These are the directories in which libraries are installed.

db

Installed Databases Directory. The directory into which record instance, template and substitutions files are installed.

Makefiles

The makefiles are described in the section Description of Makefiles.

<top>/config/*

These files contain definitions, make rules, and perl scripts used by the various Makefiles in <top>. Please note that most developers will only need to modify the files RELEASE and, until release 3.13.2, CONFIG_APP.

The following describes the files generated by makeBaseApp using either template simple or example. It is possible to create templates for makeBaseApp that act differently.

CONFIG

This is the file in which you add to or modify make variables in EPICS base. A useful definition to override is:

```
CROSS_COMPILER_TARGET_ARCHS =
```

This specifies the vxWorks architecture to build. If your site builds base for multiple target architectures but your IOCs only use a single architecture, overriding this variable with a subset of the base target architectures will save build time.

If you are using capfast you may want to add the definition:

```
DB_OPT = YES
```

CONFIG_APP

This file contains definitions for external products such as EPICS base and share. You should edit this file if you are using external products besides epics_base and share. Follow the models already in the file.

NOTE: Beginning with base release 3.13.2 the definitions that previously had to be manually added to this file to support external products will automatically be generated from the definitions in RELEASE. This is done by gnumake which executes a new script makeConfigAppInclude.pl. From 3.13.2 onwards therefor it will no longer be necessary to edit this file.

RELEASE

This file specifies the location of external products such as EPICS base. All entries in the

RELEASE file must define a full path name, either directly or using gnumake macro substitutions. The procedures for going to a new release of an external product are described later in this chapter. One step in those procedures is to edit this file. The config files created by makeBaseApp provide support for the following variables:

SUPPORT
: This convenience (optional) variable is usually set to the top of the directory tree holding support applications; succeeding definitions are usually given relative to this.

EPICS_BASE
: This variable must be defined, but usually as the last entry.

TEMPLATE_TOP
: This variable specifies the location of the template top area for makeBaseApp.

RULES.Db
: This file contains rules for building and installing database files. Databases generated from templates and/or CapFast schematics are supported.

RULES.Host
: The template file includes the `RULES.Host` from base. If you want to add rules that apply to all `Makefile.Host` files then this is the place to add the rules.

RULES.Vx
: The template file includes the `RULES.Vx` from base. If you want to add rules that apply to all `Makefile.Vx` files then this is the place to add the rules.

RULES.ioc
: This is a file containing rules for the Makefiles in the directories from which IOCs are booted.

RULES.iocBoot
: This is a file containing rules for the Makefiles in the iocBoot directory. It should not be necessary to modify this file.

RULES_ARCHS
: This file includes the `RULES_ARCHS` from base. It is seldom necessary to modify this file.

RULES_DIRS
: This file includes the `RULES_DIRS` from base. It is seldom necessary to modify this file.

RULES_TOP
: This file includes `RULES_TOP` from base. If `MASTER_IOCAPPS` is defined it also runs a utility that creates soft links to the master IOC. This feature only works if the host operating system supports soft links.

makeIocCdCommands.pl
: This is a perl script that generates a `cdCommands` file for use by IOCs. See the section Component: Directory Structure : cdCommands in this chapter for details.

makeDbDepends.pl
: This is a perl script that generates make dependencies from substitutions files.

makeConfigAppInclude.pl
: Starting with release 3.13.2 this script will be provided. It automatically generates the definitions which previously had to be added to `CONFIG_APP`. It puts the definitions into a file `CONFIG_APP_INCLUDE`.

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.

base.dbd
<app>Include.dbd

These files are used to configure database definitions for the following:

- menus
- record types
- device support
- driver support
- breakpoint tables

`base.dbd` contains definitions obtained from the base release. It contains statements like:

```
include "menuGlobal.dbd"
include "menuConvert.dbd"
include "aiRecord.dbd"
#include "aaiRecord.dbd"
...
device(ai,CONSTANT,devAiSoft,"Soft Channel")
#device(ai,CONSTANT,devAiSoftRaw,"Raw Soft Channel")
...
#driver(drvXy010)
#driver(drvVxi)
...
```

Thus it has a definition for all menus, record types, devices, and drivers supplied in EPICS base. Some record types and ALL hardware device and driver support are proceeded by the comment symbol "#". You are expected to edit this file and select the desired support routines, by removing the "#" from the relevent lines.

A version of `base.dbd` appropriate to your version of EPICS base can be obtained from `<epics_base>/templates/makeBaseApp/top/exampleApp/src.`

The file `<app>Include.dbd` is used to construct a complete list of definitions needed by the application. It must contain the line

```
include "base.dbd"
```

to incorporate all the definitions from base, followed by any definitions needed for locally built support. The name of this file is specified in the `DBDEXPAND` variable in `Makefile.Host`. Then when `gnumake` is executed, a file with the name defined by `DBDNAME` in `Makefile.Host` and having all of the include statements expanded will be installed into `<top>/dbd/`.

baseLIBOBJS

This file lists all the compiled binary object files needed for the record, device, and driver support supplied in EPICS base. Since the file is intimately related to `base.dbd`, if `base.dbd` (is, is not) used in a particular `xxxApp/src` directory, then `baseLIBOBJS` should (be, not be) used in that directory.

baseLIBOBJS contains a series of definitions as follows:

```
#LIBOBJS += $(EPICS_BASE_BIN)/aaiRecord.o
#LIBOBJS += $(EPICS_BASE_BIN)/aaoRecord.o
LIBOBJS += $(EPICS_BASE_BIN)/aiRecord.o
...
#
# Device Support
#
#LIBOBJS += $(EPICS_BASE_BIN)/devAaiCamac.o
...
#
# Driver support ANSI
#
#LIBOBJS += $(EPICS_BASE_BIN)/drvAb.o
...
```

As with the base.dbd file, some record types and ALL hardware device and driver support are proceeded by the comment symbol "#". You are expected to edit this file and select the desired support routines, by removing the '#' from the front of the appropriate lines. These edits should match those made in the base.dbd file, although no harm will usually come from uncommenting entries in baseLIBOBJS that correspond to commented−out entries in base.dbd, the effect is just to use up memory unnecessarily. The reverse is not generally true, and this also assumes that there are no references to the relevent record or device type in any database loaded by the IOCs concerned.

Makefile.Vx contains rules that will combine all support into a single module with a name given by the value of LIBNAME.

NOTE: A version of baseLIBOBJS can be obtained from

```
<epics_base>/templates/makeBaseApp/top/exampleApp/src
```

APOLOGY: It would be nice if this file could be automatically generated from the information in the expanded dbd file described above. This is not currently possible because there is no naming convention for device and driver support source files.

st.cmd

The vxWorks startup file is described in a later section.

cdCommands

This file is automatically generated within each <top>/iocBoot/iocxxx directory by gnumake executing makeIocCdCommands.pl. When this cdCommands script is called by st.cmd it generates a series of vxWorks shell string variables for use with the vxWorks cd command. For example the subsequent line in st.cmd

```
cd startup
```

will change to the directory containing the st.cmd file. The perl script used differs between releases 3.13.1

and 3.13.2. The following describes the variables generated for each release

Release 3.13.1

startup
: The full path to the directory containing the startup file.

appbin
: The full path to `<top>/bin/<ioc_arch>`

share
: If `SHARE` is defined in `<top>/config/RELEASE` this gives the full path to share

Release 3.13.2

top
: The full path to `<top>`

startup
: The full path to the directory containing the startup file.

topbin
: The full path to `<top>/bin/<ioc_arch>`

In addition, for each external `<top>` listed in `<top>/config/RELEASE` via an entry "`<PROD>=<path>`" the following variables are also created:

<prod>
: The full path `<path>` to the product if the the directory actually exists

<prod>bin
: The full path to `<path>/bin/<ioc_arch>` if this directory exists.

Thus in 3.13.2 it is easy to get to the `<prod>` and `<prod>/bin/<ioc_arch>` directories of all external products specified in `<top>/config/RELEASE`. Once at the top of a product its `db` and `dbd` subdirecties can be accessed by using relative pathnames.

3.3. Using External <top> Components

The structure of `<top>` is designed to make it easy to link to other `<top>` areas, each with it's own release schedule. The basic methods that make this possible are:

- A `<supporttop>` installs its public components into standard subdirectories. For example all executables are stored into `<supporttop>/bin/<arch>`.
- To use components from another `<supporttop>` in any `<top>` do the following:
  - Add a line to `<top>/config/RELEASE` defining the external product's location, eg:

    ```
    MYPROD = $(SUPPORT)/myprod/1-3
    ```

  - If using EPICS releases before 3.13.2, add definitions to `<top>/config/CONFIG_APP`. For example:

```
ifdef MYPROD
MYPROD_BIN = $(MYPROD)/bin/$(T_A)
USR_INCLUDES += −I$(MYPROD)/include
USER_DBDFLAGS += −I $(MYPROD)/dbd
endif
```

From release 3.13.2 onwards, executing gnumake in the config directory performs an equivalent step.

- In one or more of the <top>'s source and database directories reference components from the <supporttop> as required. These references can take the following forms:
  - #include statements used in C and C++ source code will search the <supporttop>/include directory for any installed header files
  - include statements found while expanding an <app>Include.dbd file will search the <supporttop>/dbd directory for the installed dbd files named.
  - Compiled object files from a <supporttop>/bin/<arch> directory can be linked into an xxxLib object library by adding the object filename to the definition of LIBOBJS in Makefile.Vx using statements like:

```
LIBOBJS += $(MYPROD_BIN)/myprod.o
```

- From release 3.13.2 the automatically generated cdCommands file in each iocBoot/iocxxx directory defines string constants which can be used in the st.cmd file to cd to the top and bin/<arch> directories of any <supporttop> defined in <top>/config/RELEASE. For example:

```
cd myprod
dbLoadRecords("db/myprod.db","ioc=prod1")
```

- A full description on how to use an external <supporttop> module in an application should form part of the module's documentation, which may over−ride some of the above instructions. If such documentation is not provided, ask the module maintainer why not.

3.4. Switching to a New Release of an External <top> component

The file <top>/config/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
vi config/RELEASE
     change the relevent line to point to the new release
gnumake
```

All definitions in <top>/config/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 <supporttop> 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/R3.13.1
EPICS_BASE = $(SUPPORT)/base/3-13-1-asd2
```

4. Building Components

4.1. Locating Make Rules

Most directories in a <top> which contain human editable files also contain one or more Makefiles. A Makefile normally includes a file from <top>/config. Thus the Makefile "inherits" rules and definitions from config. The files in config may in turn include files from <base>/config or from some <supporttop>/config. This technique makes it possible to share make variables and even rules across applications.

4.2. Make

Where make is used

gnumake can be executed in any subdirectory where a Makefile appears, which is almost every subdirectory. Executing gnumake in a particular directory usually causes it to descend into any subdirectories and run any Makefiles found there as well.

<top>

The most useful commands at the top level directory are:

gnumake

This rebuilds and installs everything which is not up to date.
NOTE: Executing gnumake without arguments is the same as gnumake install

gnumake clean

This can be used to save disk space by deleting the O.<arch> directories from the xxxApp areas, but does not remove any installed files from the bin, db, dbd etc. directories.

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 clean uninstall

This command removes everything created and installed by gnumake.

gnumake tar

This command makes a tar image of the entire <top> directory (excluding any CVS directories).

config

Starting with release 3.13.2 running gnumake in this directory creates the file CONFIG_APP_INCLUDE.

xxxApp

Two useful commands at this level are `gnumake` or `gnumake rebuild`, which are the same as issuing the same command in each subdirectory of `xxxApp`.

xxxsrc
xxxSrc
: Running the command `gnumake` (which is the same as executing `gnumake install`) builds and installs all out of date host and IOC components described by the files `Makefile.Host` and `Makefile.Vx`. The builds are performed in subdirectories `O.<arch>`.
It is possible to build for a single architecture via the command `gnumake <arch>`. For example, if your IOC uses an MVME167 CPU, then the build directory used is `O.mv167`, and the command is `gnumake mv167`.
Another useful command is `gnumake clean` which deletes the `O.<arch>` build directories created by make. `.<arch>` can be appended to invoke clean for a single architecture.
The command `gnumake rebuild` is the same as `gnumake clean install`.

xxxdb
xxxDb
: Executing `gnumake` in this directory generates and installs database instance files, template and substitution files from plain sources or CapFast schematics.

iocBoot
: Executing `gnumake` here is the same as issuing `gnumake` in each subdirectory of `iocBoot`.

iocxxx
: Executing `gnumake` in these directories creates their `cdCommands` files. See the section [<top>/iocBoot/iocxxx/Makefile](#) below for details.

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>
: `sun4`, `solaris`, `hp700`, `mv167`, etc. − builds named architecture only.
`host` − builds for host architecture only.

`cross` − builds for vxWorks architecture(s) only.

<action>

`clean`, `inc`, `install`, `build`, `rebuild`, `buildInstall`, `uninstall`, or `tar`
NOTE: `uninstall` and `tar` can only be specified at `<top>`

<dir>

subdirectory name

4.3. Description of Makefiles

<top>/Makefile

This makefile performs a gnumake in the xxxApp and iocBoot subdirectories. In addition it allows the top level make options uninstall and tar as described in the previous section. There is seldom need to modify this file.

<top>/xxxApp/Makefile

This makefile just executes gnumake in each `*src*`, `*Src*`, `*db*` and `*Db*` subdirectory.

<top>/xxxApp/src/Makefile.Host

The following IOC related components can be built:

Breakpoint Tables

For each breakpoint table add the definition

```
BPTS += <table name>.dbd
```

Record Support

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

```
RECTYPES += <rectype>Record.h
```

and the associated record support file `<rectype>Record.dbd` must exist.
If a `menuXXX.dbd` file is present, then add the following definition:

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

Expanded Database Definition File

Files containing database definition files are expanded by utility dbExpand and installed into `<top>/dbd`. The following variables control the process:

```
DBDEXPAND += xxxInclude.dbd
DBDNAME = xxxApp.dbd
USER_DBDFLAGS += −I <include path>
USER_DBDFLAGS += −S <macro substitutions>
```

```
DBDINSTALL += xxx.dbd
```

where the entries are:

DBDEXPAND
: A list of files containing database definitions to be expanded.

DBDNAME
: The name of the output file to contain the expanded definitions which will be installed into `<top>/dbd`.

USER_DBDFLAGS
: Flags for dbExpand. Currently only an include path and macro substitution are supported.

DBDINSTALL
: Installs the named files into `<top>/dbd` without expansion.

`Makefile.Host` has many facilities for building host components. Definitions given below containing <`arch`> can be used to provide settings for use when building for a specific host architecture, and the <arch> part of the name should be replaced by the architecture concerned, e.g. `solaris`, `hp700` etc. If a `_DEFAULT` setting is given but a particular <`arch`> requires that the default not be used and the required setting is blank, the value "`−nil−`" should be assigned to the relevent <`arch`> variable definition.

Products to be built	
PROD	Product names (without execution suffix) to build and install (e.g. `PROD=myprod`)
SRCS	Source files needed by every PROD (e.g. `SRCS=a.c b.c c.c`)
<prod>_SRCS	Source files needed to build a specific PROD (e.g. `myprod_SRCS=a.c b.c c.c`)
PROD_<arch>	os specific products to build and install
<prod>_SRCS_<arch>	os specific source files to build a specific PROD
PROD_DEFAULT	products to build and install for systems with no PROD_<arch> specified
<prod>_SRCS_DEFAULT	source files needed to build a specific PROD for systems with no <prod>_SRCS_<arch> specified
Building libraries	
LIBRARY	Name of library to build. The name should NOT include a prefix or extension, e.g. specify `Ca` to build `libCa.a` on Unix, `Ca.lib`, `CaObj.lib` or `Ca.dll` on WIN32
LIBSRCS	Source files for building LIBRARY (e.g. `LIBSRCS=la.c lb.c lc.c`)
LIBSRCS_<arch>	os−specific library source files

LIBSRCS_DEFAULT	Library source files for systems with no LIBSRCS_<arch> specified
SHARED_LIBRARIES	Build shared libraries? Must be `YES` or `NO`
SHARED_LIBRARIES_<arch>	Build shared libraries on <arch>? Must be `YES` or `NO`
SHARED_LIBRARIES_DEFAULT	Build shared libraries for os systems with no SHARED_LIBRARIES_<arch> specified
SHRLIB_VERSION	Shared library version number
Compiler flags	
USR_CFLAGS	C compiler flags for all systems
USR_CFLAGS_<arch>	os−specific C compiler flags
USR_CFLAGS_DEFAULT	C compiler flags for systems with no USR_CFLAGS_<arch> specified
<prod>_CFLAGS	prod specific C compiler flags (e.g. `xxxRecord_CFLAGS=-g`)
USR_CXXFLAGS	C++ compiler flags for all systems
USR_CXXFLAGS_<arch>	os−specific C++ compiler flags
USR_CXXFLAGS_DEFAULT	C++ compiler flags for systems with no USR_CXXFLAGS_<arch> specified
<prod>_CXXFLAGS	prod specific C++ compiler flags
USR_CPPFLAGS	C pre−processor flags (for all makefile compiles)
USR_CPPFLAGS_<arch>	os specific cpp flags
USR_CPPFLAGS_DEFAULT	cpp flags for systems with no USR_CPPFLAGS_<arch> specified
<prod>_CPPFLAGS	prod specific C pre−processor flags (e.g. `xxxRecord_CPPFLAGS=-DDEBUG`)
USR_INCLUDES	Directories to search for include files with `-I` prefix (e.g. `-I$(EPICS_EXTENSIONS_INCLUDE)`)
<prod>_INCLUDES	Directories to search for include files when building a specific product(e.g. `-I$(MOTIF_INC)`)
Linker options	
USR_LDFLAGS	linker options (for all makefile links)
USR_LDFLAGS_<arch>	os specific linker options (for all makefile links)
USR_LDFLAGS_DEFAULT	linker options for systems with no USR_LDFLAGS_<arch> specified
<prod>_LDFLAGS	prod specific ld flags
USR_LIBS	load libraries (e.g. `-lXt -lX11`) (for all makefile links)
USR_LIBS_<arch>	os specific load libraries (for all makefile links)

USR_LIBS_DEFAULT	load libraries for systems with no USR_LIBS_<arch> specified
<prod>_LIBS	prod specific ld libraries (e.g. `probe_LIBS=X11 Xt`)
PROD_LIBS	libs needed to link every PROD for all systems
PROD_LIBS_<arch>	os−specific libs needed to link every PROD
PROD_LIBS_DEFAULT	libs needed to link every PROD for systems with no PROD_LIBS_<arch> specified
<lib>_DIR	Directory to search for the specified lib. (For libs listed in PROD_LIBS, <prod>_LIBS and USR_LIBS)
SYS_PROD_LIBS	system libs needed to link every PROD for all systems
SYS_PROD_LIBS_<arch>	os−specific system libs needed to link every PROD
SYS_PROD_LIBS_DEFAULT	system libs needed to link every PROD for systems with no SYS_PROD_LIBS_<arch> specified
<prod>_SYS_LIBS	prod specific system ld libraries (e.g. `m`)
Header files to be installed	
INC	List of include files to install into `$(INSTALL_DIR)/include`
INC_<arch>	os specific includes to installed under `$(INSTALL_DIR)/include/os/<arch>`
INC_DEFAULT	include files to install where no INC_<arch> is specified
Perl, csh, tcl etc. script installation	
SCRIPTS	scripts to install for all systems
SCRIPTS_<arch>	os−specific scripts to install
SCRIPTS_DEFAULT	scripts to install for systems with no SCRIPTS_<arch> specified
TCLLIBNAME	List of tcl scripts to install into `$(INSTALL_DIR)/lib/<arch>` (Unix hosts only)
TCLINDEX	Name of tcl index file to create from TCLLIBNAME scripts
Test programs	
TESTPROD	Product names (without execution suffix) to build but not install. Built from source file having same name.
TESTPROD_SRCS	List of source files needed to build every TESTPROD
Documentation	
MAN1 MAN2 MAN3 etc.	List of man files to be installed into relevent `$(INSTALL_DIR)/man/man?` subdirectory
DOCS	List of text files to be installed into the `$(INSTALL_DIR)/doc` directory

HTMLS_DIR	Hypertext directory to be created as `$(INSTALL_DIR)/html/$(HTMLS_DIR)`
HTMLS	List of files to be installed into the `$(INSTALL_DIR)/html/$(HTMLS_DIR)` directory
TEMPLATES_DIR	Template directory to be created as `$(INSTALL_DIR)/templates/$(TEMPLATE_DIR)`
TEMPLATES	List of template files to be installed into `$(TEMPLATE_DIR)`
Options for other programs	
YACCOPT	yacc options
LEXOPT	lex options
SNCFLAGS	snc options
E2DB_FLAGS	e2db options
SCH2EDIF_FLAGS	sch2edif options
RANLIBFLAGS	ranlib options
Miscellaneous settings	
CMPLR	C compiler selection, `TRAD`, `ANSI` or `STRICT` (default is `STRICT`)
INSTALL_LOCATION	Location of install directory (default `$(TOP)`)
USER_VPATH	List of directories that gnumake should search for source files not in the current directory
HOST_WARN	Are compiler warning messages desired (`YES` or `NO`) (default is `NO`)
HOST_OPT	Is compiler optimization desired (default is `NO` optimization)
STATIC_BUILD	Is static build desired (`YES` or `NO`) (default is `NO`)
Facilities for building Java programs	
CLASSES	Names of Java classes to be built and installed
TESTCLASSES	Names of Java classes to be built
PACKAGE	Names of Java package to be installed
JAR	Name of Jar file to be built
JAR_INPUT	Names of files to be included in JAR
MANIFEST	Name of manifest file for JAR
Facilities for Windows 95/NT resource (`.rc`) files	
RCS	Resource files needed to build every PROD
<prod>_RCS	Resource files needed to build a specific PROD
<prod>_RCS_<arch>	os specific resource files to build a specific PROD

<top>/xxxApp/src/Makefile.Vx

Note that the rules for `Makefile.Vx` are older and subtly different to those for `Makefile.Host`. The following components can be built:

Record Support
For each new record type, the following definitions must be added to the makefile

```
LIBOBJS += <rectype>Record.o
```

and both source files `<rectype>Record.c` and `<rectype>Record.dbd` must exist.

Device, Driver, other C modules
For each such module, add a definition:

```
LIBOBJS += <name>.o
```

All such files will be combined into the library specified by LIBNAME. It is also possible to generate object files not placed in LIBNAME by using either of the definitions:

```
PROD += <name>.o
TARGETS += <name>.o
```

Both will cause the specified file to be generated, but PROD will also install the generated file into <top>/bin/<target_arch>.

Library files
A file containing all LIBOBJS is installed into <top>/bin/<arch> with the name specified by LIBNAME. For example:

```
LIBNAME = xxxLib
```

State Notation Programs
For each state notation program, add the definition:

```
LIBOBJS += <name>.o
```

The state notation programs must be called `<name>.st`.

Scripts, etc.
A definition of the form:

```
SCRIPTS += <name>
```

results in file `<name>` being installed from the src directory to the `<top>/bin/<arch>` directory.

`iocCore` and `seq`
In order to have iocCore and seq in the bin directory where the standard st.cmd file expects to find them, the following must appear:

```
INSTALLS += iocCore seq
```

NOTE: The above line only needs to appear in one application within each <ioctop>.

Other definitions:

USR_CFLAGS	C compiler flags
<prod>_CFLAGS	product specific C compiler flags
USR_CXXFLAGS	C++ compiler flags
<prod>_CXXFLAGS	product specific C++ compiler flags
CPPFLAGS	cpp flags
<prod>_CPPFLAGS	product specific cpp flags
USR_INCLUDES	Include directory (e.g. `-I$(EPICS_EXTENSIONS_BIN)`)
USR_LDFLAGS	linker options
<prod>_LDFLAGS	product specific ld flags
INC	header files to install
BIN_INSTALLS	Files in any directory to install to `$(INSTALL_BIN)`
YACCOPT	yacc options
LEXOPT	lex options
SNCFLAGS	snc options
<prod>_SNCFLAGS	product specific state notation language flags
VX_WARN	Compiler warning messages desired (`YES` or `NO`) (default `NO`)
VX_OPT	Is compiler optimization desired (`YES` or `NO`) (default is `NO` optimization)
INSTALL_LOCATION	Installation directory (defaults to `$(TOP)`)

<top>/xxxApp/xxxDb/Makefile.Host

This makefile creates and installs databases and related files.

Databases

Supported are:

- Plain databases
  - from dct/gdct/editor generated source files (`*.db`)
  - from CapFast schematics (`*.sch`)
- Template generated databases
  - template database from dct/gdct/editor generated source file (`*.template`)

- template database from CapFast schematic (`*.sch`)
- substitutions file from editor generated source (`*.substitutions`)
- substitutions file generated by a script (e.g. querying a relational database)

- Database optimization using the `dbst` tool (i.e. removing all fields with default values)

For template generated databases either the fully inflated database or the template and substitutions files may be installed (so the IOC may load the database using `dbLoadRecords()` or `dbLoadTemplate()`).

For all these types of databases, the names of the files to install have to be specified. Make will figure out how to generate these files:

`DB += xxx.db`
: Generates `xxx.db` depending on which source files exist. If `xxx.db` is template generated, the inflated database will be installed.

`DB += xxx.template xxx.substitutions`
: Generates and installs these files, i.e. the database must be inflated on the IOC using `dbLoadTemplate()`.

In order to record dependency information correctly all template files that are needed but not installed (i.e. those not listed in `DB`), must be added to the `USES_TEMPLATE` variable:

```
USES_TEMPLATE += yyy.template
USES_TEMPLATE += $(SHARE)/installDb/zzz.template
```

If specified with a path (full or relative), the templates will be soft linked (UNIX) or copied (WIN) into the `O.<arch>` directory. After the first make run, template dependencies will be generated automatically.

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.

NOTE: 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.

Other definitions:

E2DB_FLAGS	e2db options
SCH2EDIF_FLAGS	sch2edif options

Related Files

Expanded Database Definition File
: Files containing database definition files are expanded by utility dbExpand and installed into `<top>/dbd`. The following variables control the process:

```
DBDEXPAND += xxxInclude.dbd
DBDNAME = xxxApp.dbd
USER_DBDFLAGS += -I <include path>
```

```
USER_DBDFLAGS += -S <macro substitutions>

DBDINSTALL += xxx.dbd
```

where the entries are:

DBDEXPAND
: A list of files containing database definitions to be expanded.

DBDNAME
: The name of the output file to contain the expanded definitions which will be installed into `<top>/dbd`.

USER_DBDFLAGS
: Flags for dbExpand. Currently only an include path and macro substitution are supported.

DBDINSTALL
: Installs the named files into `<top>/dbd` without expansion.

Breakpoint Tables
: For each breakpoint table add the following definition

```
BPTS += <table name>.dbd
```

<top>/iocBoot/Makefile

This executes gnumake in each iocxxx subdirectory.

<top>/iocBoot/iocxxx/Makefile

This makefile has a rule to generate the `cdCommands` file. Make sure that the definition:

```
ARCH = <arch>
```

refers to the correct architecture for your IOC processor.

Since `cdCommands` is generated, the user created and/or modified files can be independent of location, using the `cdCommands` definitions to provide path names where needed. See 3.2. Directory Structure: cdCommands for details.

5. CVS Reference

The CVS utility is used to put all user editable files under source/release control. This section gives a brief description of the commands normally used by application developers. Consult the CVS manual for more details.

CVSROOT

Your environment variable `CVSROOT` should point to the CVS repository for IOC Applications. The following command displays the location of `CVSROOT`:

```
echo $CVSROOT
```

For example, at APS/ASD the command should show:

```
/usr/local/iocapps/cvsroot
```

At APS all `<top>` applications are stored under `$CVSROOT/iocsys`. Ask your EPICS system manager about the use of CVS at your site.

Commands

This section gives a brief description of the CVS commands. Wherever <filename> is shown a list of filenames is allowed. If <filename> is not specified then most commands apply to the entire directory and all subdirectories.

A useful option for `cvs` is:

```
cvs –n <command>
```

This will execute any command to demonstrate what it would do without actually making any changes to the current directory or the repository.

help

Typing

```
cvs help
```

gives overall cvs help.

checkout

To check out an entire `<top>` area issue the commands:

```
cd <anywhere>
cvs checkout iocsys/<top>
```

This will retrieve the latest version of `iocsys/<top>` into `<anywhere>/iocsys/<top>`

watch

Files (or complete directory trees including and entire <top> area) can have a watch placed on them. When a watch is placed on a directory `cvs` creates working copies read only. Users must execute a cvs edit command to obtain a read/write file. Facilities are provided to list all people editing a file and to be sent an e−mail message whenever someone executes the cve edit or commit commands for a watched file. Read the CVS manual for details.

edit

If you want to edit a file and it is read only because a watch is in effect then execute the command:

```
cvs edit <filename>
```

unedit

If you have started editing a file and decide to abandon your changes or not make any changes issue the command:

```
cvs unedit <filename>
```

add

The command:

```
cvs add <filename>
```

places a directory or file under CVS control. This command must be given for each directory and file to be added to the repository. A subsequent commit command will also be required.

remove

The command:

```
cvs rm <filename>
```

removes the specified file from the repository. The file is not actually deleted but is moved to the "attic". Thus previous versions can still be retrieved.

diff

The command:

```
cvs diff <filename>
```

compares the working copy of the file with the latest version in the repository.

The diff command has options that allow you to see the differences between any two versions committed to the repository.

update

The command:

```
cvs update −d <filename>
```

brings the development area into sync with the latest versions committed to the repository. A message starting with U is given for each file or directory that is updated. If the message starts with M the file named has been modified in this directory. A message starting with the letter C means that a conflict was found between the working file and a change in the

repository. Conflicts must be resolved manually (edit the file and look for regions bounded by <<<<<< ====== and >>>>>> characters).

The option `-d` means add any new subdirectories that have been created in the repository.

commit

The command:

```
cvs commit <filename>
```

commits changes to the repository. You are asked for comments via your favorite editor.

status

The command:

```
cvs status <filename>
```

shows the status of the file. The −v option shows all tag information for the file.

log

The command:

```
cvs log <filename>
```

displays the commit messages for all versions of the specified file.

tag

The command:

```
cvs tag <official release name>
```

is used by the Application System Manager to tag official application releases.

import

This command is used to put an existing tree of files into the `cvs` repository. Assume that a developer has created a new directory tree for a new `<top>` application in a directory `newapp`. It can be imported into the repository via the command:

```
cvs import −m "Creating" iocsys/newapp newapp start
```

`.cvsignore` file

Any directory can contain a file with the name `.cvsignore`. It contains a list of file and directory names and filename patterns that should be ignored by CVS. For example all generated directories and files should be listed in `.cvsignore`.

`.cvsrc` file

CVS will use a file called `.cvsrc` in your home directory to specify default options to its commands. The following settings are strongly recommended for use by all users. Note that the gap between the CVS command name and the option letters must be a tab character, not just spaces:

```
checkout    -P
update    -d -P
export    -kv
```

The `-P` options to checkout and update cause CVS to remove any empty directories. Update's `-d` option tells it to create any directories that have been added to the repository since the last update. The `-kv` flags to export cause it to replace any RCS keywords (e.g. `$Id$`) with strings which will not be changed if the exported source code is later imported into a different repository.

6. Creating <top> Applications

6.1. makeBaseApp

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

- <top>/Makefile
- <top>/config − Build configuration subdirectory and associated files
- <top>/xxxApp − A set of directories and associated files for a major sub−module.
- <top>/iocBoot − A subdirectory and associated files.
- <top>/iocBoot/iocxxx − A subdirectory and files for a single ioc.

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

Usage

makeBaseApp has three possible forms of command line:

<base>/bin/<arch>/makeBaseApp.pl −l [options]
: List the application templates available. This invocation does not alter the current directory.

<base>/bin/<arch>/makeBaseApp.pl [−t type] [options] app ...
: Create application directories.

<base>/bin/<arch>/makeBaseApp.pl −i [options] ioc ...
: Create ioc boot directories.

Options for all command forms:

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

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

−d
: Verbose output (useful for debugging)

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

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

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

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

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

-a <arch>
: Set the IOC architecture (e.g. mv167). If not specified you will be prompted for this information. For use with the -i invocation only.

Environment Variables:

EPICS_MBA_DEF_APP_TYPE
: Application type you want to use as default

EPICS_MBA_TEMPLATE_TOP
: Template top directory

Description

To create a new <top> issue the commands:

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

makeBaseApp does the following:

- EPICS_BASE is located by checking the following in order:
  - If the -b option is specified it is used.
  - If a <top>/config/RELEASE file exists and defines a value for EPICS_BASE it is used.
  - It is obtained from the invocation of makeBaseApp. For this to work, the full path name to the makeBaseApp.pl script in the EPICS base release you are using must be given.
- TEMPLATE_TOP is located in a similar fashion:
  - If the -T option is specified it is used.
  - If a <top>/config/RELEASE file exists and defines a value for TEMPLATE_TOP it is used.
  - If EPICS_MBA_TEMPLATE_TOP is defined it is used.
  - It is set equal to <epics_base>/templates/makeBaseApp/top
- If -l is specified the list of application types is listed and makeBaseApp terminates.

- If `-i` is specified and `-a` is not then the user is prompted for the IOC architecture.
- The application type is determined by checking the following in order:
    - If `-t` is specified it is used.
    - If `EPICS_MBA_DEF_APP_TYPE` is defined it is used.
    - If a template `defaultApp` exists, the application type is set equal to `default`.
    - If a template `exampleApp` exists, the application type is set equal to `example`.
- If the application type is not found in `TEMPLATE_TOP`, makeBaseApp issues an error and terminates.
- If `Makefile` does not exist, it is created.
- If directory `config` does not exist, it is created and populated with all the config files.
- If `-i` is specified:
    - If directory `iocBoot` does not exist, it is created and the files from the template boot directory are copied into it.
    - For each `<ioc>` specified on the command line a directory `iocBoot/ioc<ioc>` is created and populated with the files from the template (with `ReplaceLine()` tag replacement, see below).
- If `-i` is NOT specified:
    - For each `<app>` specified on the command line a directory `<app>App` is created and populated with the directory tree from the template (with `ReplaceLine()` tag replacement, see below).

Tag Replacement within a Template

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

- Two perl subroutines are defined within makeBaseApp:
    - ReplaceFilename − This substitutes for the following in names of any file taken from the templates.
        - `_APPNAME_`
        - `_APPTYPE_`
    - ReplaceLine − This substitutes for the following in each line of each file taken from the templates:
        - `_USER_`
        - `_EPICS_BASE_`
        - `_ARCH_`
        - `_APPNAME_`
        - `_APPTYPE_`
        - `_TEMPLATE_TOP_`
        - `_IOC_`
- If the application type directory has a file named `Replace.pl`, it can:

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

6.2. Application Templates Supplied With Base

EPICS base supplies the following sets of template files

- exampleApp
- exampleBoot
- simpleApp
- simpleBoot

simpleApp creates an xxxApp with a `Db` and `src` directory. Each directory contains skeleton makefiles. simpleBoot creates an `iocBoot` directory and `iocBoot/iocxxx` directories. Each directory contains makefiles. The `iocxxx` directories also contain a skeleton `st.cmd` file.

6.3. Example Application

exampleApp and exampleBoot create a complete example application. They contain the following files.

```
<app>App
    src/
        Makefile
        Makefile.Host
        Makefile.Vx
        caExample.c
        sncExample.st
        xxxRecord.dbd
        xxxRecord.c
        devXxxSoft.c
        <app>Include.dbd
        base.dbd
        baseLIBOBJS
    Db/
        Makefile
        Makefile.Host
        dbExample1.db
        dbExample2.template
        dbExample2.substitutions

iocBoot/
    Makefile
    nfsCommands
    ioc<app>
        Makefile
        st.cmd
```

caExample.c

A Host application that interfaces to Channel Access. It is executed from a Unix shell by issuing the command:

```
caExample "pvname"
```

It issues a Channel Access get request for the specified process variable and prints the value. If you have booted an IOC from the example then try the following:

- On the IOC console type the command:

  ```
  dbl
  ```

  This produces a list of all the records the IOC contains.
- On the host system change to the directory: <top>/bin/<host_arch>/
- Execute the command:

  ```
  caExample "pvname"
  ```

  where pvname is one of the record names shown by `dbl`

dbExample1.db

This is a file containing some example record instances. Each name starts with `$(user)`, which will be expanded into the login name of the person who executed makeBaseApp at database load time. The records are:

`$(user)aiExample`
: A passive ai (analog input) record which obtains its input from record `$(user)calcExample`.

`$(user)calcExample`
: This is a calc (calculation) record that acts as a counter that continually counts from 0 to 9. It is scanned once a second. It also has a forward link to `$(user)aiExample`. Since the latter is passive it will also scan once a second.

`$(user)xxxExample`
: This is a sample record of type xxx, as described below. It is a passive record. You can change its VAL field from a Channel Access client or by using the IOC command `dbpf`.

dbExample2.template
dbExample2.substitutions

This is another example of record instances. The record instance file is generated from a template, which is instantiated using the entries in the substitutions file. Each record name in the template file starts with `$(USER)`, which is replaced by the login name of the person

who executed makeBaseApp when the template is instantiated at build time. Each line in the substitutions file creates a two record database, whose records are described above (`aiExample` and `calcExample`).

sncExample.st

This is a state notation language example for compilation by the state notation compiler and execution on the IOC by sequencer. It prints a message on the IOC console every time the VAL field of the record `<user>xxxExample` transits through the value 5.0. This time the <user> part of this record name is instantiated when the template is created by makeBaseApp.

xxxRecord.dbd xxxRecord.c

A skeleton record support module. The record support module is the one described in the Application Developer's Guide.

devXxxSoft.c

A device support module for xxxRecord. The device support module provides synchronous support for the record support.

6.4. st.cmd

This file is the vxWorks startup file. The version created by makeBaseApp contains:

```
# Example vxWorks startup file
#Following must be added for many board support packages
#cd <full path to iocxxx>
< cdCommands

#< nfsCommands

cd appbin
ld < iocCore
ld < seq
ld < exampleLib

cd startup
dbLoadDatabase("../../dbd/exampleApp.dbd")
dbLoadRecords("../../db/dbExample1.db","user=<user>")
dbLoadRecords("../../db/dbExample2.db")

iocInit
seq &snctest
```

The commands dbLoadDatabase, dbExpand, dbLoadRecords and dbLoadTemplate are described in the "Database Definition" chapter of the Application Developer's Guide.

The cdCommands file is created when gnumake is run, and defines several vxWorks variables for use with cd later on in the startup file. Definitions are provided for:

- appbin − A full path name to <top>/bin/<target_arch>
- startup − A full path name to <top>/iocBoot/iocxxx
- share − A full path name to share if SHARE is defined in <top>/config/RELEASE

NOTE: From release 3.13.2 the name appbin becomes topbin. Other definitions are also created by cdCommands. See the description of <top>/iocBoot/iocxxx/Makefile in section 4.3 for details.

The first ld command loads the core EPICS components. The files iocCore, seq and exampleLib are installed when gnumake is run in the <top>/xxxApp/src directory. exampleLib contains the executable for all record, device, and driver support as well as any other application specific object modules. If an IOC wants to use support generated in a sub−application src directory, this statement will have to be changed to coincide with the LIBNAME value.

The dbLoadDatabase command loads the definitions of all menus, record types, device support, driver support, and breakpoint tables needed in this IOC. These are actually expanded files created by dbExpand and installed into dbd. If an application wants to use database definitions generated in a sub−application src directory, this statement will have to be changed to coincide with the DBDNAME value.

The command:

```
dbLoadRecords("../../db/dbExample1.db","user=<user>")
```

is an example command for loading record instances using macro substitution at load time. One of these commands is supplied for each record instance file. Note that the <user> parameter in the example will have been replaced by the login name of the person running makeBaseApp.

The iocInit command initializes the EPICS system.

The seq command shows how to start a sequence (state notation language) program.

7. APS/ASD Configuration Management Procedures

7.1. Overview

This section describes a set of procedures for managing ioc software. EVERYTHING is for the sole purpose of supporting iocs, thus it does not discuss management of high level applications. The procedures are intended for an operational facility. During commissioning activities some more relaxed procedures may be appropriate.

The Configuration Management process described here relies on a certain level of understanding and responsibility by the engineers who implement it, and may not always rigorously maintain a previous "known working" set of software online for an IOC to be rolled back to in the event of problems while making changes. It is always possible to retrieve the last version from the system backup tapes, but this may not be acceptable for some high−availability applications. Sites desiring more beaurocratic but "safer" procedures may wish to contact JLAB or SNS to find out about their approaches.

The key features of the APS Configuration Management System are:

- CVS
  All human created/edited files are put in the cvs repository dedicated to ioc applications
- `<top>`
  The complete set of ioc software is divided into `<top>` areas. Each `<top>` area is managed independently. Each follows its own development and release schedule.
- `<ioctop>`
  This is a `<top>` dedicated to a set of iocs that are booted from `<ioctop>/iocBoot/iocxxx`
- `<supporttop>`
  A `<top>` area that contains products shared across multiple `<ioctops>`.
- Releases
  Each `<top>` can have multiple releases.
- Operations and Development areas
  Releases of `<top>` appear in an operations tree. Operational iocs are booted from this tree. Developers can checkout any `<top>` in a private tree.
- `<supporttop>` releases are not modified in the operations tree.
  Because a `<supporttop>` is used by multiple `<ioctops>` it should never be modified in the operations tree. Any modifications require a new release. It may be appropriate to relax this rule during a period of commissioning.
- `<ioctop>` releases may be modified in the operations tree.
  This makes it easy to make small ongoing changes to operational systems.

Background

Before EPICS base release 3.13, APS/ASD used an Application Source Release control system called appSR. appSR uses SCCS for source file control. Since appSR was written several things that impact S/R control have changed.

- With release 3.13 the method of configuring record/device/driver support changed.
- Both EPICS base and extensions have switched from sccs to `cvs`.
- `base/config`, i.e. definitions and rules for GNU make, has evolved so that is now possible to create makefiles that are much simpler, more powerful, and extendable.

For the above reasons we decided to redo the APS/ASD IOC Applications S/R control system. Our goals were:

- SCCS will no longer be used.
- All functionality performed by appSR will be done with a combination of CVS and GNU make.

A major decision was *What directory structure should we use*. Before the original appSR was written a **lot** of discussion went into this topic. There were several meetings between the Application Developers and Bob Zieman, who implemented appSR. This directory structure was used to implement the entire APS/ASD contol system software. Thus a lot of correct decisions were made. The only major problem is that it was not easy to share code, mainly record/device/driver support, across <`top`> areas. This resulted in sharing by copying source modules from one <`top`> area to another. When one developer would make changes to a source module, the other developers would often not even be aware of the changes. Thus over time the source modules evolved in different directions.

Since the overall directory layout appears to be correct, it is kept in the new system. A brief description of the APS/ASD environment may help explain why the directory layout works.

- APS accelerator
    - The APS accelerator complex is composed of four major subsystems: Linac, PAR (Positron Accumulator Ring), Booster Syncrotron, and Storage Ring. Each subsystem is controlled by a separate set of IOCs. Thus it is entirely appropriate to have separate <`top`> areas for each subsystem.
    - Each subsystem can be viewed as a separate set of applications, e.g. RF, Magnet Power Supplies, Diagnostics, etc.
- IOC responsibilities.
    - A particular application may be spread over multiple IOCs. For example the control for storage ring applications is normally spread over many IOCs just because of the physical size of the storage ring.
    - A particular IOC may contain parts of multiple applications. For example vacuum and power supplies normally share IOCs.
- Application Developers.
    - For the Linac, a single Application Developer has final responsibility for all controls applications, i.e. RF, vacuum, Diagnostics, etc.
    - For the par, booster, and sr, Application Developers are assigned application areas. For example the same person is responsible for almost all diagnostic controls for the par, booster, and storage ring.

One other topic to discuss before describing the overall directory structure is the *idealized* Application

Development Cycle. It consists of the following steps:

- Define I/O requirements.
  This involves meeting with the user, in this case the engineers who are responsible for the application in order to decide the types and number of I/O modules needed.
- Assemble control hardware and software.
  This includes IOCs, I/O modules, software device/driver support, etc. If EPICS software support is not already available, it has to be written and tested.
- Build databases, sequence programs, etc.

In reality there is overlap between these steps. In addition as new needs arise the three steps again have to be performed. However, an application developer tends to spend a large part of his/her attention on each step before moving on to the next step.

7.2. CVS Repository for IOC Applications

The cvs tree is arranged in two parts:

```
$CVSROOT/
    support/
        <supporttop1>/
        <supporttop2>/
        ...
    ioc/
        <ioctop1>/
        <ioctop2>/
        ...
```

So at APS/ASD the repository tree is:

```
$CVSROOT/
    support/
        base/
        allenBradley/
        mpf/
        share/
        bitBus/
        radMon/
        switchgear/
        runcontrol/
        motorTransform/
        saverestore/
        PSCU/
        ...
    ioc/
        sr/
        mcr1/
        runcontrol/
        tune/
        s40misc/
        srbpm/
        ...
    ...
```

7.3. Adding a new <top> to the CVS repository

This section provides guidelines for creating a new <top> application that will be maintained in the local cvs repository, i.e. it does NOT apply to <top> applications maintained somewhere else.

Assume you have a directory tree starting at <working>/<newtop> that you want to put in the cvs repository at location `$CVSROOT/<system>/<newtop>`Just issue the commands:

```
cd <working>/<newtop>
gnumake clean uninstall
cvs import −m "Creating" <system>/<newtop> <newtop> start
```

where `<system>` is either `support` or `ioc`.

A `.cvsignore` file should appear in all `<top>` areas and also in all `<top>/iocBoot/ioxxxx` directories.

`<top>/.cvsignore` should contain:

```
bin
include
lib
man
dbd
templates
doc
db
javalib
html
```

`<top>/iocBoot/iocxxx/.cvsignore` should contain:

```
cdCommands
```

`.cvsignore` files are added to the repository just like any other file.

Documentation and Release Notes

Each <top> should provide some documentation describing the application and also release notes describing each release. The release notes should mention any dependencies on other <top> applications it uses. The recommended documentation format, at least for release notes, is HTML. These documents can be put in any directory containing a Makefile.Host. Adding the lines:

```
HTMLS += xxx.html
HTMLS_DIR = .
```

to this Makefile will result in xxx.html being installed into the directory `$(INSTALL_LOCATION)/html/.`, where `$(INSTALL_LOCATION)` is normally <top> The `HTMLS_DIR` value can be used to target a subdirectory under `$(INSTALL_LOCATION)/html`, useful for longer documents with many sections and graphics files.

7.4. Operations Directory Structure

An operations tree has the structure:

```
<operations>/
    RX.XX.X/
        support/
            <supporttop1>/
                1-1/
                ...
            <supporttop2>/
                1-1/
                ...
            ...
        ioc/
            <ioctop1>/
                1/
                ...
            <ioctop2>/
                1/
    RY.YY.Y/
        ...
```

NOTES

- A new `RY.YY.Y` directory will be needed whenever a significantly changed version of base is needed which requires that all ioc and support applications be rebuilt to use the new version. Some examples of such major changes are:
    - A new version of vxWorks is being used.
    - The underlying epics structures are modified.
    - The set of fields in dbCommon are changed.
- `<supporttop>/x-y` releases are created via `cvs export` commands because it is not permissible to modify an operational <supporttop>.
- `<ioctop>/x` releases are created with a `cvs checkout` command because it is permissible to modify an operational ioc release. See below for reasons.

If only bug fixes are being applied to EPICS base then a new `RX.XX.X` directory is generally not needed. Instead a new `support/base/x-y-site` release can be created.

A portion of the actual APS/ASD operations area is:

```
/usr/local/iocapps/
R3.13.1/
    support/
        base/
            3-13-1-asd1/  This is actually 3.13.1 with bug fixes
        share/
            1-3/          What is left after unbundling
        allenBradley/
            1-1/
```

```
            mpf/
                1-3/
            bitBus/
                1-1/
            radMon/
                1-1/
            switchgear/
                1-1/
            runcontrol/
                1-1/
            motorTransform/
                1-1/
            saverestore/
                1-1/
            PSCU/
                1-1/
        ioc/
            sr/
                1/
            mcr1/
                1/
            runcontrol/
                1/
            tune/
                1/
            s40misc/
                1/
            srbpm/
                1/
            others as they get converted to 3.13.1
```

7.5. Support Management

7.5.1 Overview

A <supporttop> must follow a few simple rules in order to be used by an <ioctop>.

- If it uses EPICS_BASE or another <supporttop>, it must have a config/RELEASE file.
- It must install everything intended for use by an <ioctop> into a directory immediately under its <top>.

Thus it must have the following directory structure:

```
<top>
    config/
        RELEASE
    bin/
        <arch>/
        ...
    lib/
        <arch>/
```

```
    ...
dbd/
db/
include/
```

NOTE: Most <supporttop>s will need only a subset of `bin`, `lib`, `dbd`, `db` and `include`.

At least the following two types of support areas may be used by an <ioctop>

- A <supporttop> that is developed in the same cvs repository as the <ioctops>
- A <supporttop> that follows the above directory structure but is maintained in a different cvs repository. Examples are epics base and epics unbundled products such as the allenBradley support. To allow local bug fixes to be made to any of these remotely obtained products, the following management procedure is used:

    - The product is exported from the cvs repository where it is maintained, or otherwise copied into a local directory area.
    - This area is then imported into the local cvs repository as a vendor import.

    NOTE: EPICS base will follow this procedure. This makes it easy to make local bug fixes rather than having to create and use a completely new EPICS release for even minor changes.

In addition an application might use commercial products. Since these products are not under our control, each <ioctop> has to decide how to use the product. Wherever possible, however, the product location should be defined by a line in the <ioctop>/config/RELEASE file.

7.5.2. Procedures for <supporttop> Maintained in Local Repository

Summary:

- Developers develop in a private area checked out from the main cvs branch.
- When a new version is ready for operations, all changes are committed and a release tag is created.
- Operations creates an exported version of the new version in the operational area.
- If it is necessary to patch an older release, a branch is created based on that release and the patch applied to the new branch.

Developer's Cookbook

General Guidelines:

- Have a standard place to do development. At APS/ASD it is recommended that each developer use the following paths, which should be created with `mkdir` commands:

    ```
    <home>/iocapps/RX.XX.X/support
    ```

- Checkout ONLY the <supporttop> applications for which you are responsible.

- If `cvs watch on` has been set then all files will be checked out read only. If you want to edit a file, first issue the command:

```
cvs edit <filename>
```

  DO NOT just issue a chmod to change the file permissions.
- Try to do all development on the main cvs branch. It should seldom be necessary to create branches for <supporttops>. If it is get help unless you understand the procedures described below. This section only discusses changes on the main branch.
- If a new application is being created get help unless you understand the procedures described previously.

Initial Checkout:

A private working version of <supporttop> is checked out via the commands:

```
cd <support>
cvs checkout -d <supporttop> support/<supporttop>
cd <supporttop>
    all development done here
```

Tagging a new release:

When changes have been made and it is time for operational use, all changes should be committed and a new release tag created:

```
cd <support>/<supporttop>
    make sure all changes have been committed
cvs tag Rx-y
```

Tags should take the form `Rx-y`, where `x` is a major release number for the module and `y` is a minor release number. For example the first release is `R1-1`, the next minor release `R1-2`, etc.

You can see all previous tags by issuing the commands:

```
cd <support>/<supporttop>
cvs status -v Makefile
```

After tagging the new release you should notify operations to install it in the operations tree.

Creating a tar file:

If you also want to create a tar file of containing the new release for use elsewhere, you should **not** just tar up your working directory, but instead use the following commands:

```
cd <somewhere>
cvs export -kv -d <supporttop>-x.y -r Rx-y support/<supporttop>
tar cf <supporttop>-x.y.tar <supporttop>-x.y
```

```
gzip <supporttop>-x.y.tar
```

Note that the naming conventions used for the export directory and tar file above (a hyphen `-' separates the name and release number, and the release number has a period `.' between major and minor parts) are those used for most open source software. The `-kv` option to `cvs export` instructs it to expand RCS keywords so that they will be preserved without further change if the resulting files are ever imported into a different CVS repository, thus preserving their history.

Patching Old Releases

NOTE: This should seldom be necessary.

If it becomes unavoidable to apply patches to an old release instead of upgrading to a newer version, a branch must be created in the repository to hold these changes. The following procedure demonstrates this:

```
cvs rtag -b -r Rx-y Bx-y support/<supporttop>
cd <somewhere>
cvs checkout -d <supporttop>/Bx-y -r Bx-y support/<supporttop>
cd <supporttop>Bx-y
    make, test and commit changes
cvs tag Rx-y-1
```

The `cvs rtag -b` can be issued from anywhere as it creates the branch tag `Bx-y` by working directly on the files in the repository. The checkout command specifying `-r Bx-y` then retrieves that new branch. All subsequent cvs commands issued in the new area will apply to the branch rather than the versions on the main trunk. The command `cvs tag Rx-y-1` makes a tag for modified version on this branch. A good convention to follow is that branch releases should have the same tag as the release with `-i` appended, where `i` represents the patch release number for this branch. Thus the first patch is `Rx-y-1`, the second `Rx-y-2` etc.

Operations Cookbook

Managing local <supporttop> modules

Installing a new release for use in the Operations Area

Operations installs a new <supporttop> by exporting the new release into the operational directory tree:

```
cd <operations>/RX.XX.X/support/<supporttop>
cvs -r export -d x-y -r Rx-y support/<supporttop>
cd x-y
gnumake
```

Note that the operations version is created using `cvs export` rather than `cvs checkout`. This means that no CVS directories appear in the operations version of a `<supporttop>`. This directory area should never be modified once operational `<ioctops>` start using it. If a modification is required a new (possibly

patch) release must be made using the methods described in this document.

Installing a patch release to the operations area

Patch releases are installed in the operations tree exactly like any other release, using the commands:

```
cd <operations>/RX.XX.X/support/<supporttop>
cvs -r export -d x-y-i -r Rx-y-i support/<supporttop>
cd x-y-i
gnumake
```

7.5.3. Managing a <supporttop> from another Repository

This applies to products like EPICS base, allenBradley, mpf, etc. The basic rules are:

- The first time a product is put in the local repository:
    - It is imported into `support/<product>` giving a vendor tag reflecting the product and a release tag reflecting the release version.
    - Any local modifications that are needed (e.g. changes to config, NOT bug fixes) are committed on the main branch.
- Any local bug fixes are made on a new branch, NOT the main branch.
- When a new release of the product from the maintainer is to be incorporated:
    - It is imported again, using the same vendor tag but with a release tag reflecting the new maintainer's release version number.
    - Any local changes that occurred on the main branch between the two import operations will automatically be merged into the new maintainer's release and the result placed at the head of the main branch.
    - Any conflicts due to overlapping changes between the newly imported release and local modifications will be highlighted to be resolved manually.

If possible the local release tags assigned to products maintained elsewhere should identify the imported release. For example epics base would be imported with a command like:

```
cvs import -m "import base 3.13.1" support/base base release3-13-1
```

After making and committing any local configuration changes this would be given a tag of `R3-13-1`. If it is necessary to patch this release of base, the first patch tag would be `R3-13-1-asd1`.

The following examples are for mpf. Assume that the first version of mpf is version 1-1 and the second version is 2-1.

Importing for the first time

Assume that a tar file containing version 1-1 of mpf has been extracted into the directory `<working>/mpf`

```
cd <working>/mpf
cvs import −m "Import mpf version 1−1" support/mpf mpf release1−1
cd <working>
/bin/rm −rf mpf
```

Now checkout the new product from the main branch, make local changes, tag it, and delete the working copy.

```
cd <working>
cvs checkout −d mpf support/mpf
cd mpf
    make and test local configuration changes
cvs commit −m "Configuration changes for release 1−1"
cvs tag R1−1
cd <working>
/bin/rm −rf mpf
```

Only changes that are needed for the local environment should be made here, for example the `config/RELEASE` file should be modified. Bug fixes should NOT be applied here − see below for how to make bug fixes. The local release tag `R1−1` matches the mpf version number, indicating that the software is identical to that of the official mpf release but with local configuration steps fully made.

Importing a new version

Assume that version 2−1 of mpf has been extracted from a tar file into `<working>/mpf`:

```
cd <working>/mpf
cvs import −m "Import mpf version 2−1" support/mpf mpf release2−1
    The import will indicate any conflicts that need resolving.
cd <working>
/bin/rm −rf mpf
cvs checkout −d mpf −j release1−1 −j release2−1 support/mpf
cd mpf
    Make changes to resolve conflicts and add any new local configuration
    changes needed.  The following commit is needed to remove files that have
    not been locally modified and which are not present in the new release.
cvs commit −m "Merged local changes with mpf 2−1"
cvs tag R2−1
cd <working>
/bin/rm −rf mpf
```

The `cvs import` command will report any conflicts that have to be merged by hand, and provides instructions on how to do this if it is necessary. In our example this was the case, and the subsequent `cvs checkout` command merged the changes between the two vendor releases onto the main branch.

Applying Local Patches

If it is necessary to make local patches to software that is maintained elsewhere, the changes must be made on a side branch. If the changes were made on the main branch then importing later versions of the product that had these changes incorporated would flag these up as conflicts that had to be resolved manually. The disadvantage is that local bugfixes which have not been included in an official release will have to be re−applied on another new branch, but this can be regarded as an incentive to ensuring that the official

maintainer be sent copies of all bug fixes found.

Assume that it is necessary to apply a patch to version 1−1 of mpf:

```
cvs rtag −b −r R1−1 B1−1 support/mpf
cd <working>
cvs checkout −d mpf −r B1−1 support/mpf
cd mpf
    make any necessary changes and commit them
cd <working>/mpf
cvs tag R1−1−asd1
cd <working>
```

The new release tag that operations should install is `R1−1−asd1`.

Exporting a new release to the Operations Area

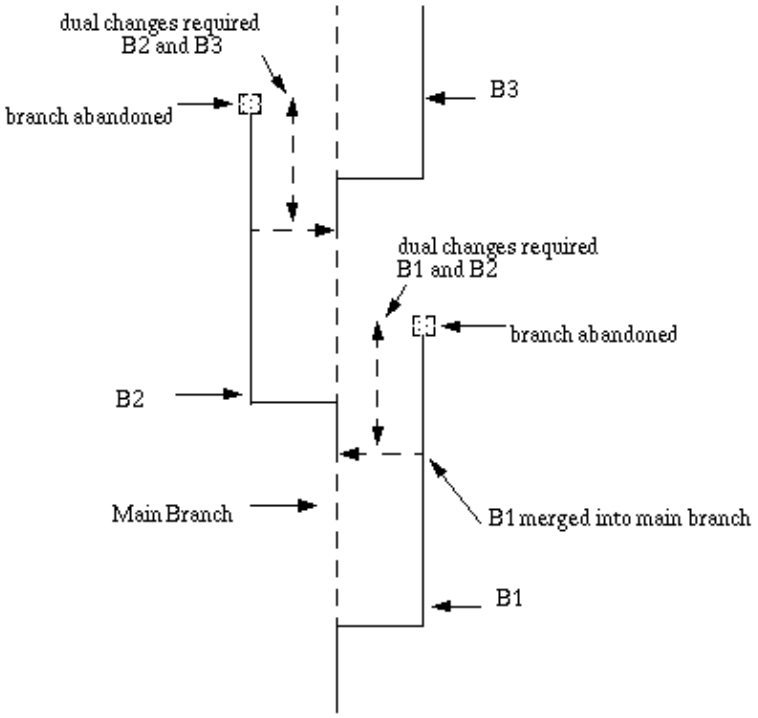
Releases are prepared for operations just like for a <supporttop> being maintained in the local repository.

7.6. IOC Management

7.6.1. Overview

The procedures for <ioctop>s are designed so that simple changes to the operational tree are easy. It is expected that there will be frequent (weekly or even daily) small changes such as adding database records, changing HOPR, LOPR values, etc. All these minor changes are made on a side branch rather than the main branch of the cvs tree. Developers check out a private copy of this branch. The operational area, from which the iocs are booted, has the same branch checked out. Changes appear in the operational area by a developer committing changes and operations performing cvs updates.

The following diagram shows how an `<ioctop>` evolves over time.



A new branch is needed when:

- It is not possible to boot all iocs from the same directory structure, for example while converting to a new release of EPICS base.
- Major changes are being made to `<ioctop>` and it must be possible to quickly revert to the old working system.

Assume the current operational system is

```
<operations>/R3.13.1/ioc/<ioctop>/1
```

Each `<ioctop>` application developer has a private area

```
<somewhere>/ioc/<ioctop>/1
```

Both of these areas have been checked out via the commands

```
cd <somewhere>/ioc/<ioctop>
cvs checkout -d 1 -r B1 ioc/<ioctop>
cd 1
```

All cvs commands issued in an `<ioctop>/1` area will be made to the B1 cvs branch. Until it is time for a new release no changes are made to the main cvs branch.

Assume that a new release of epics is available, which requires a new side branch for `<ioctop>`. The following steps are taken by whoever is preparing it for the new epics release:

Step 1

The main branch is checked out of the repository and the B1 changes merged back into the main branch.

```
cd <somewhere>
cvs checkout -d <ioctop> ioc/<ioctop>
cd <ioctop>
cvs update -d -j B1
cvs commit -m "B1 merged into main branch"
```

This is shown in the diagram as "B1 merged into main branch".

Step 2

Changes can be made to `<ioctop>` and committed without affecting the B1 branch. During this step the major development work needed is done on the main cvs branch.

Step 3

When it is ready for use by other application developer's and/or operations, a new side branch B2 is created. The R2 tag also created below marks the file versions at the root of the new branch for later reference.

```
cd <somewhere>/<ioctop>
cvs commit -m "changes for new epics release"
cvs tag R2
cvs tag -b B2
cd <somewhere>
/bin/rm -rf <ioctop>
```

Step 4

The new area is checked out via the commands:

```
cd <somewhere>/<ioctop>
cvs checkout -d 2 -r B2 ioc/<ioctop>
```

Changes can be made independently on either branch B1 or B2, and iocs can be booted from either area. Once all iocs are successfully using B2, the B1 cvs branch, operations 1 area, and any 1 private areas are "abandoned", i.e. they no longer need to be maintained. The abandoned branch ends in a hollow box in the diagram.

IMPORTANT WARNING. While B1 and B2 are both active, it is the application developer and operations responsibility to make any changes to both releases. This starts from the time step 2 is performed until B1 is no longer active. The diagram identifies this period with "dual changes". CVS can help with this task, but in most cases it will probably be easier and just as quick to make and commit the same change in both areas.

The diagram shows that the same steps are followed for branch B3.

Why is minor development done on a branch?

Development is done on a branch so that when a new branch becomes the operational branch, nothing has to be done to get the operational tree on the latest release − it is already using it!

If development were done on the main cvs branch, then when new development got to the point where it was desired to make this the new operations release, a new release would have to be generated on a side branch and the official area switched to use the new release. In practice at APS this time has often only been known once the system was already operational. The only easy way to prove that the new release is really ready is to run it in an operational mode. By then however, operations staff are already saying: "It works, please leave it alone."

Versions of a branch

`<ioctop>` developers may want to create a tag on a branch before and after making extensive changes to the branch or before merging a branch back onto the main tree. These tags should always be of the form Rx−y where x is the branch and y is a minor release number. This if the current branch is B2 then the tags are R2−1, R2−2, ... These tags are NOT created for the purpose of creating operational releases. They are only to tag snapshots of the cvs repository at key moments.

7.6.2. Developer's Cookbook

General Guidelines:

- Have a standard place to do development. Each site may establish guidelines. At APS/ASD it is recommended that each developer work in an area under their home directory as follows:

  ```
  <home>/iocapps/RX.XX.X/ioc/
  ```

- Checkout ONLY the `<ioctop>`s for which you are responsible.
- If `cvs watch on` has been set then all files will be checked out read only. If you want to edit a file issue the command:

  ```
  cvs edit <filename>
  ```

  DO NOT just issue a chmod to change the file permissions.
- Do all development on the correct cvs branch. If it is time to create a new cvs branch for an <ioctop> get help unless you understand the procedures described previously.
- Be careful. Remember that the operations area is using the same cvs repository as you.
- Whenever possible use the same config/RELEASE definitions as operations.

Check Out an <ioctop> branch

```
cd <ioc>/<ioctop>
cvs checkout −d x −r Bx ioc/<ioctop>
cd x
    all development is done here
```

where x is the branch desired. For example if B1 is the current branch than x is 1.

Committing Changes

Be Careful. Remember that when operations performs updates and builds in the operations tree the operational system will be using your modifications. During operational periods only changes really needed for operations should be committed.

```
cd <ioc>/<ioctop>/x
    commit ONLY the files actually changed
```

Now prepare instructions for what operations should do. The instructions should specify cvs updates and makes at the lowest level possible. Do NOT just give instructions to go to `<ioctop>/x` and update and rebuild.

7.6.3. Operations Cookbook

Ask for detailed instructions from developer's about where to perform cvs updates and makes.

7.7. Additional Guidelines for Operations

Backup each <ioctop>

Because `<ioctop>`s can be modified, it is possible that a developer could perform cvs commits that will break an operational system when a cvs update is done in the operations area. In order to prevent a major problem, procedures should be in place to backup all `<ioctop>`s on a regular schedule when the facility is on an operational period. Procedures should also be in place to restore a particular `<ioctop>` from the backup.

Application developers must be aware that only minor changes should be made during operational periods. When a developer makes a change commits should be made only to the necessary files rather than commits at the top level. He/she should instruct operations about what should be updated and rebuilt, i.e. update just the files necessary and only build the components affected by the updates.

Updates in <ioctop> areas

Because application developers can accidentally commit changes that would break the operational system care should be taken when performing cvs updates on an `<ioctop>` in the operations area. A good idea is to do the following before any updates:

Wherever you plan to issue a cvs update first issue the command

```
cvs -n update
```

which will list the files that are liable to be changed. Then on any files you want to update issue the command

```
cvs diff -r Bx <filename>
```

where `Bx` is the current branch tag, to see the modifications that have been committed. Only if you are satisfied with these changes should you perform the actual update.

$CVSROOT/CVSROOT/cvsignore

This file should contain

```
O.*
```

which makes cvs ignore all O.* files .

8. EPICS CVS Repository

8.1. Overview

All development of EPICS core software makes use of a central CVS repository at APS. Remote access to this is possible but is controlled by the requirements of the APS firewall and DoE policies on computer security. The remainder of this chapter assumes that the user has some experience in using CVS.

Licensing and Distribution

Until the question of future EPICS licensing is fully resolved the source code for the core software can only be made available to collaboration members, thus anonymous CVS read access to the whole repository cannot be provided. Anonymous access to the `epics/modules` directory containing unbundled hardware support packages is possible though, and experiments are being made in providing this facility from the APS web−site. A link to this facility will appear on the APS EPICS home page if the experiments prove successful.

CVSROOT

The EPICS CVS repository is at `/net/phoebus/epicsmgr/cvsroot` on any of the machines in the APS controls group network. Users must be a member of the Unix group `epicsmgr` to be able to access the repository using CVS.

8.2. Base

EPICS base is found at `epics/base` relative to the CVSROOT. Note that `base` is also defined as the CVS module name for this subdirectory.

Files are tagged with each release, and distributed versions also have a branch created to allow bug fixes to be applied and retrieved without also getting changes that are intended for the next major release. For example the branch with tag `epics_R3_13_1_branch` is used for all bug fixes to version 3.13.1 but not include modifications being developed for version 3.14 (which should appear only on the main branch until released).

Marty Kraimer <mrk@aps.anl.gov> should usually be consulted before checking in changes to base. Janet B. Anderson <jba@aps.anl.gov> is in charge of the release process.

8.3. Modules

Hardware support code (device and driver support and some specialized record types) is being unbundled from base, and the intention is for 3.14 to not have any hardware−specific code in it. The existing support routines will be packaged up as `<supporttop>` applications and these will be placed into the `epics/modules` area of the repository. Responsibility for future maintenance of these modules is being divested from APS where other volunteers can be found, and after unbundling any remaining modules will be left fallow. Sites interested in taking on a such a module are asked to contact Andrew Johnson

<anj@aps.anl.gov> about this. The list of modules and Module Owners will be merged into Steve Lewis' List of EPICS Supported Hardware pages in due course.

Module Classifications

The repository structure within `epics/modules` will reflect the classification of the module concerned. The basic philosophy behind the repository structure is that the location of a particular module is determined by its functionality, not by the connection method needed to control it. Cross−references will obviously be needed from the particular bus involved, but these should be given in the documentation and web−site for the bus, not the repository. Some devices may have more than one method of connection, and in these cases the name of the <supporttop> directory should include the bus used (for example `epics/modules/bus/gpib/mpfGpib` is the module that supports GPIB communications over MPF). The module categories are as follows:

Bus interface *(`epics/modules/bus`)*
: This classification is for bus interface and device support software such as GPIB, CAMAC, Bitbus, IPAC and MPF etc. These modules provide the bus interface and generic device support, but not software specific to a particular remote I/O modules which connects to that bus; that software lives in one of the other areas below (unless it's an adapter to a different bus type). It is up to the module owner to decide whether to have a monolithic <supporttop> which might have to contain several drivers for different interfaces to the same bus type, or to split these into a common bus−related module and separate support applications for each interface driver.

PLC interface *(`epics/modules/plc`)*
: Interfaces that communicate with a PLC using some software protocol are collected here. The precise distinction between plc and bus interface depends whether the device being communicated with is usually end−user programmable or not, thus the Allen−Bradley scanner support appears in the Bus interfaces, but if the the Allen−Bradley PLC−5 interface using the DF−1 serial protocol is ever managed centrally it should be installed here.

Serial interfacing *(`epics/modules/serial`)*
: Serial is really another example of a bus type, but being pragmatic about it there are so many different interfaces to serial devices it was felt necessary to give them a top−level structure. Distinction between a PLC interface and a Serial one should use the 'is it programmable?' test.

Soft support *(`epics/modules/soft`)*
: Support applications which don't have associated hardware. The unbundled SNL sequencer will be distributed as a <supporttop> application here, as will the pal record, and the vxWorks statistics and symbol device supports.

Analog I/O *(`epics/modules/analog`)*
: This classification is for analog and waveform I/O device and driver support. Initially there will be a <supporttop> for each manufacturer, although the module owner may further subdivide this into more <supporttop> applications if desired. These <supporttop>s may have dependencies on one of the Bus interfaces above.

Digital I/O *(`epics/modules/digital`)*
: Digital I/O device support appears here, including long integers. The same substructure applies as described for Analog I/O.

Motor control *(`epics/modules/motor`)*
: The stepperMotor record is here, also intended for DC motor controllers etc. A single <supporttop> is provided for each manufacturer containing the device and driver support

interface software, to encourage the development of a single driver interface for use by both the stepperMotor and motor record types.

Timing I/O *(`epics/modules/timing`)*
Pulse generators, counters, timing and event interfaces belong here. The event distribution system record types are in the common application, and the APS−developed hardware support for these is in aps. The pulseCounter, pulseDelay and pulseTrain records that are specific to the Mizar MZ8310 are with the associated device and driver support in the mizar application.

Instrument I/O *(`epics/modules/instrument`)*
Anything that supports a number of related I/O variables or has a complex record type which doesn't obviously fit into one of the other categories may be placed here. Each instrument manufacturer should have their own subdirectory, and the support applications for the individual instruments will exist below that.

Site−specific *(`epics/modules/site`)*
Sites that share custom hardware with one or more other labs can place support modules here if they don't fit any of the other categories particularly well. Any substructure should be decided by the lab concerned.

Additional categories or sub−divisions can be added if they prove necessary. For example some of the client extension programs could be converted to using the makeBaseApp structure, and it would be appropriate to give them in a suitable classification in this area rather than the existing monolithic `epics/extensions` area.

The above path names may seem to result in a very broad tree, putting only a very small number of devices/drivers in each <supporttop> application. This is deliberate, as at most sites an individual <ioctop> application will usually only need a reasonably small number of device types. Experience at APS has shown that bundling too much support code into a single application produces a software maintenance nightmare further down the line, and this is one reason why we are unbundling the hardware support. A broad tree should not add significantly to the tasks an application developer needs to perform, and EPICS site managers can help by including the necessary additional lines (commented out as desired) in the `config/RELEASE` file of the local makeBaseApp templates.

When installing packages for use by applications, we recommend that the same relative path name be used as given above, but the particular version of the module will need to be given a name which reflects the particular module version. At APS the release number (such as 2−1) is used as the last component of the path name to the module, so that for example the Allen Bradley Scanner <supporttop> might be installed at `/usr/local/iocapps/R3.14.0/bus/allenBradley/2−1`. This makes it possible to install a new version of a <supporttop> application at any time but still ensure that only the application engineer in charge of a particular IOC can decide when to switch to using the new version. The APS Procedures section of this document describes this approach, although it does not use the classification structure described here as it was developed for earlier releases of EPICS.

The preferred method of distributing modules is to provide a tar file for each release, accessed through the module's web−site or ftp server. Module owners are also encouraged to read the Linux Software Release Practice HOWTO document which gives good advice on release procedures.

Any hardware record, device or driver support which was provided with release 3.13.1 of EPICS base but which is missing from the above list will be deleted. Should you need any such software you should check on

Steve Lewis' Supported Hardware web page and approach the current maintainer directly, or ask on tech−talk. As a last resort you can always obtain the source code from release 3.13.1 and create your own support application for it.

8.4. Module Owner Responsibilities

A significant part of the work of a Module Owner (MO) is to maintain the code, fixing reported bugs or merging bug fixes from other users. However equally important or even more vital are the tasks associated with creating and maintaining documentation so the module can be used, and with packaging up releases and making the software easily available to other EPICS sites. It may be appropriate for some modules to divide the work between several people, some of whom might not be software developers (web−site maintenance for example).

Documentation

One of the first jobs should be to develop a web−site for the module. Eventually this should contain:

- A list of the hardware manufacturers and models supported by the module. Links to their web−sites would be useful.
- Instructions for obtaining and/or links to versions of the source code (see Distribution below).
- Information on the version(s) of base supported and any other modules which will be needed.
- Documentation on how to install this module at a new site.
- Documentation on how to use this module in an <ioctop> application.
- Release notes describing any changes between different release versions.
- A list of sites known to be using this module (this provides some indication of software quality).

Ideally copies of the documentation should be distributed with the software, but at the very least there should be a README file present which gives the URL for the module's web−site and the name of the Module Owner.

A template module web−page has been developed (using Netscape Composer, although any HTML editor can be used) which MOs may start from if they so wish. This will obviously need to be modified to suit particular circumstances by changing or removing parts as necessary (search for underscore characters in the HTML file to find all the parts that will need replacing). If you develop your own site, please try to ensure that all the above information can be accessed by the users somehow, although it does not have to be all on one page.

If you do not have access to a web−server or would prefer to maintain the web−site at a central location, APS will make an area of its web−site available to nominated MOs for module use. Requests for web−space and an account on the APS server should be made to Bill McDowell.

Distribution

The recommended method is to provide a tar file for each documented release of the <supporttop>, accessible via the module's web−site or an ftp server. This is pretty much an industry−standard approach for obtaining software, and as long as there are no copyright problems in making the software available on a public server

it should be the preferred method.

Unless there are particular reasons not to do so, try and keep older versions of the software available. This particularly applies where a particular version of EPICS base is required. MOs may be called upon to incorporate bug fixes into old module versions and generate new releases of these after an upgrade of EPICS base has required a significant change to the software. The release version number should indicate the relevant parentage.

We are looking at the possibility of providing a mirrored copy of the CVS repository outside the APS firewall, providing anonymous read−only access. This should make it easier for sites to obtain and install upgrades without compromising security for the code−base. Until the legal copyright issue has been resolved this would only serve the $CVSROOT/epics/modules area.

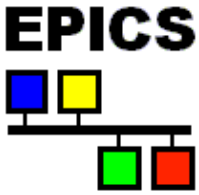
Development

As changes occur to EPICS base, modifications may be needed to the <supporttop> applications to remain compatible with the new versions. The changes needed and any tools to help with this will be published by the base developers as soon as these have been confirmed, and any modifications thought to require significant effort should have been discussed on tech−talk or other forum beforehand. The MO will be responsible for ensuring that the required changes are applied and tested, and a new release created.

Delegation

There may be several developers in the EPICS consortium with an interest in a particular module, and the MO is at liberty to distribute the workload associated with maintaining the module as she/he sees fit. Overall responsibility for the software and final authority for what should or should not be included in it will remain with the MO however.

EPICS _template_ Module



Module Owner:
_owner_name_

This page is the home of the EPICS _template_ module, which provides _record/device/driver_ support for _hardware_. This site gives access to the software source code, information on other modules which are needed to install and run it, and documentation on the how to include and use it in your EPICS applications. Please email any comments and bug reports to _owner_name_ who is responsible for coordinating development and releases. _peer_name1_ and _peer_name2_ are also familiar with this module and may be willing to help with some problems.

Hardware Supported

The module can drive the following types of hardware:

- _manufacturer1__model1_ _description1_
- _manufacturer2_ _model2_ _description2_

Where to Find it

You can download the software by anonymous ftp from the _lab_ ftp site at ftp://_ftp_site_/_directory_, or directly from the links in the table below:

Module Version	EPICS Release	Filename
_1-1_	R3.13.1	_module1-0.tar.gz_
_2-0_	R3.14.0	_module2-1.tar.gz_

Required Modules

Applications using this software will also need these other modules to be installed:

template Version	Requires module	Release needed
_1-1_	_required_	_versions_
_2-0_	_required1_	_versions_
required2	_versions_	

Site Installation and Building

After obtaining a copy of the distribution, it must be installed and built for use at your site. These steps only need to be performed once for the site (unless versions of the module running under different releases of EPICS and/or the other required modules are needed).

1. Create an installation directory for the module, usually this will end with /support/_class_/_module_.
2. Unpacking the distribution tar file produces a <supporttop> directory named after the release number.
3. Edit the config/RELEASE file and set the paths to your installation of EPICS and to the _required1_ and _required2_ modules.
4. Run `gnumake` in the top level directory and check for any compilation errors.
5. Please email _module_owner_ so s/he can keep track of which sites are using this software.

Application Installation

To use the installed and built support software in an <ioctop> application, make the following changes to the application:

1. Edit the `config/RELEASE` file and add the line

```
_TEMPLATE_=/path/to/module/version
```

2. See the documentation for the _requires1_ and _requires2_ modules for instructions on how to install these in the application. An <ioctop> application must use the same version of these modules that the _template_ module has been built with.
3. Edit the config/CONFIG_APP file and add the lines

```
ifdef _TEMPLATE_
USR_INCLUDES += -I$(_TEMPLATE_)/include
_TEMPLATE_BIN = $(_TEMPLATE_)/bin/$(T_A)
USER_DBDFLAGS += -I $(_TEMPLATE_)/dbd
endif
```

4. In the application source directory where the base object files are linked together, edit Makefile.Vx and add

```
LIBOJBS += $(_TEMPLATE_)/_templateLib_
```

5. Ensure the following is included in the construction of the application's .dbd file

```
include "dev_template_.dbd"
```

6. Rebuild the application and use the newly installed support as desired.

Documentation

The following documentation is available:

- Introduction to the _template_ module (HTML) (pdf − 200 Kb) (PostScript − 150 Kb)
- Release notes (HTML)

In Use

This software was originally developed by/for _developer_ at _lab_ and is used at the following EPICS sites:

- _lab1_
- _lab2_ _division1_
- _lab2_ _division2_

_website_maintainer_