UNIX DEVELOPMENT ENVIRONMENT

Principles of Design

13th September, 2003
Last Revised: 17 March 2005
Greg White, Michael Zelazny, Kristi Luchini
SLAC, Stanford University, California, USA

Revision History

Date	Revision	Description	Author
09/23/03	1.0	Initial Version	Greg White
10/27/03	1.1	Added diagrams for main dirs and release	Greg White
11/06/03	1.2	Added CVS for IOC development	Greg White
11/19/03	1.3	Added Makefile framework description	Greg White
11/23/03	1.4	Added Makefile development and testing	Greg White
11/23/03	1.41	Moved Release System Table to BUG	Greg White
12/15/03	1.5	Added adding new Functional File Types	Greg White
3/17/04	1.6	Added Process Management Chapter	Greg White
4/15/04	1.7	Added Process Management Scripts	Greg White
3/14/05	1.8	Added Distribution system	James Silva

Reference:

See also the Unix Development Environment, Basic Users Guide. That document describes the design of the mechanisms and tools described in this document.

Modifying this file:

This file is located in $CD_SOFT/html/unix/dev/ug/POD.doc (http://www.slac.Stanford.edu/grp/cd/soft/unix/dev/ug/POD.doc). It is on the web at http://www.slac.Stanford.edu/grp/cd/soft/unix/dev/ug/POD.pdf

When modifying this file, please also create the pdf version and put it in the same directory.

1. Production and Development Hosts

This chapter deals with the distinction between computers (hosts) on which we develop software, and those on which we run it to control the accelerator.

1.1. The Development System

The Development System is basically the set of "Taylored"[1] machines such as tersk, flora, slcs5 and so on, on which software is written, debugged and version controlled. The development machines' file system is AFS, and most are Sun Microsystems machines running the Solaris operating system. Once developed, the binary and necessary supporting files, such as scripts, are moved to the Production System.

1.2. The Production System

The Production System is the set of Unix machines which run the operational aspects of the accelerator complex. These machines use the NFS filesystem, but like the Development System, they are too mostly Sun Microsystems running Solaris.

Call the host on which software is run for the operational accelerator a "Production host". The production hosts of most EPICS related host software are called "gateways" (see 1.4). The production host of IOC software is the IOC on which that software will be run.

1.3. "prodondev"

At present, some significant body of software for the control system is presently run from Taylored (AFS) machines which are also used for Development, for instance gateways 4 and 5, which run 8-pack and NLCDEV software, and DM displays. Aida running on slcs6 will be another example. So, those machines are in some sense Development Hosts which run production software – "production on development", or "production on Taylor", we call that software "prodondev".

1.4. "gateways"

In this document, the term Production Host is used, where frequently the term "gateway" may be more familiar. The word gateway has been avoided here though partly because its definition isn't clear – it's roughly understood to mean a host running the gateway EPICS CA proxy server for purposes of network isolation - and partly because we do have hosts running production software which are not gateways in this narrow sense.

Table 1: Taxonomy of control system software "execution mode"

Function	"Execution Mode"	Where is this hosted
"dev" – software which is in a development directory on a development system host.	DEV	Development (AFS) hosts
"production on development" – software which is in a production directory on a development control system host, and is being used operationally to run the accelerator.	PRODONDEV	Development (AFS) Hosts
"prod" – software which is in a production directory on a production control system host.	PROD	Gateways (NFS)

Note, if the AFS system, or our connection to it, goes down, "production on development" software will not be available to run the accelerator complex.

[1] Taylor is a remote computer configuration utility developed at SLAC for unix system administration.

1.5. Nomenclature

This section describes some especially pedantic distinctions we make and the phrases we use to label them.

1.5.1. “on dev”, “in dev”, “on prod”, “in prod” and “prodondev”

A potential source of confusion is that, in the software development environment, both Development and Production machines contain software at both the development and production stages of release. This is so that development software can be tested on Production machines – making use of production databases and production networks and so on, and so that production software can be exercised on a development host - where it is less invasive to operations. This leads to the phrases “on dev”, meaning on a development machine, “in dev” meaning in a development directory, “on production”, meaning on a production machine, and “in production”, meaning in a production directory (though not necessarily *on* production)[2].

[2] Note that “prodondev” does not merely mean “on dev in prod”, it additionally says that the software is used to run the accelerator even though it is only on a dev machine.

2. Primary Directories of the Unix Environment

This chapter describes the filesystems and important top-level directories used in the Unix control system environments. First, we describe in what way the development and production hosts are the same in this respect, and then describe each in more detail, including the important differences.

2.1. Two File-systems, one directory organization

Files on the development system are in the AFS file-system, rooted at /afs/slac/g/cd/soft/. Files on the production systems are on the NFS file-system, rooted at /usr/local/cd/.

Table 2: Production and Development Directory Tree Roots

Host Type	Root directory	Environment Variable
Development	/afs/slac/g/cd/soft/	$CD_SOFT
Production	/usr/local/cd/soft/	$CD_SOFT

Since EPICS software in general requires many files in support of each application to be available on production, but that software is developed on development systems, there must be an organizing principle used for the directory structure on production. The principle we use is that we keep the released files on production in a directory structure which largely mirrors the released file-system directories on development. In fact, the way the automatic deployment system described later works, is to export directories from release directories on the development hosts, over to same named directories on production hosts. That is, beneath $CD_SOFT you will find a similar file-system structure for executable software, on both development and production machines. All the software development is done on development machines, so the directories for source code, and build output are only on Development (see Table 3).

Table 3: Directory structures under CD_SOFT on Dev and Prod. The table illustrates schematically the dispersal of files in the main directories of released software. Source files are in CVS and the "reference" directories. The reference directories also contain build object files. Executables are delivered to release directories, and are then escalated through from ref, to dev, new and finally prod. All built executables are left in ref, and accumulate in prod. Only dev, new and prod are distributed to production machines.

DEVELOPMENT (AFS, eg flora)	PRODUCTION (NFS, eg gateways)
$CD_SOFT/cvs/ gui/ common/ app/ (All source)	Not applicable
$CD_SOFT/ref/ gui/.../O.solaris common/.../O.solaris app/.../O.solaris (All source and object) $CD_SOFT/tst/ disp/ script/ sun4-solaris2/ (All executables)	Not applicable
$CD_SOFT/dev/ disp/ script/	$CD_SOFT/dev/ disp/ script/

DEVELOPMENT (AFS, eg flora)	PRODUCTION (NFS, eg gateways)
sun4-solaris2/	sun4-solaris2/
$CD_SOFT/new/ disp/ sun4-solaris2/	$CD_SOFT/new/ disp/ sun4-solaris2/
$CD_SOFT/prod/ disp/ sun4-solaris2/ All executables	$CD_SOFT/prod/ disp/ sun4-solaris2/ All gateway executables
$CD_SOFT/bck/ disp/ sun4-solaris2/ All executables	Not applicable

2.2. Primary Directories of Development System

The directories employed in the Development System are described here. These are rooted at $CD_SOFT, which is set to /afs/slac/g/cd/soft/. This is summarized in Table 4.

2.2.1. CVS and the CVS Reference Directory (cvs/ and ref/)

The ref/[3] directory contains a read-only reference version of all the file in the CVS, repository db, which is itself under cvs/. Additionally, ref/ contains both the "object" directories (like O.solaris), and the executable (or "install") directories (like sun4-solaris/bin) which result from building the source directories. This is so that we could write the basic makefiles for doing builds of the software in such as way that they could build software in a developer's working directory, or in ref/, without modification. In this way, ref/ is just the "mega-project" directory, and behaves identically to a developer's project.

2.2.2. src/ and tst/

In order to help distinguish the CVSed source code from build output, there are two additional directories under $CD_SOFT, src/ and tst/; src/ contains symlinks to the source code directories under ref/ (i.e. gui/, common/ etc.), and tst/ contains symlinks to the executable, or "build products" directories built from what is in src/, eg disp/, solaris/, script/ etc.).

2.2.3. dev/, new/ and prod/

These are the so called "release directories". They implement a system of software release escalation, in which new software is successively moved from dev/ to new/ to prod/ after different levels of testing. They can be thought of as "alpha", "beta" and "production" release. The makefile system described below implements moving them software from one to the next in a well organized way.

[3] In the following, directories under $CD_SOFT may be referred to without the $CD_SOFT/ prefix. For instance, ref/ refers to $CD_SOFT/ref.

They each contain the same subdirectories – the install directories of the makefile build output. Software must use makefiles developed from the scheme described below to use this automatic release escalation.

Directory	Important Subdirectories	Function
cvs/	See Chapter 3.4	**The CVS repository** The version controlled db of all critical source files used to build the control system.
ref/	See Chapter 3.4	**The CVS "reference area"** A readable version of each file in the CVS repository, plus the result of gmaking those files. Executables such as scripts or displays should not be run from directories under ref/, they must be run from their **release** directory (see below).
lib/		Archive (non-dynamic) libs (.a), these are not escalated.
{tst, dev, new, prod }	 script/ @sys/lib/ @sys/bin/ @sys/pbin/ javalib/ disp/ matlab/ python/ ora/ include/	**The Release Directories** Platform independent non-executable scripts. Platform dependent object libraries. Platform dependent executable. Platform dependent executable of system software. java class and jar files Epics .dl files Matlab executable and .m files Python scripts Oracle files, SQL etc Include files shared among applications
ioc/	<epics-version>/@	IOC applications software. Symlink to ../ref/epics/<version>/

Table 4 Important top level directories on Development systems

2.2.4. ioc/

ioc/ contains the build output from host side builds,of EPICS ioc software, and symlinks to the cvs reference directories for that software. Presently, the makefiles for this ioc software do not use the system of building and release escalation that host side software does, so the release esclation directories (dev/ new/ prod/ do not apply to it).

2.3. Primary Directories on Production Systems

The system of directories on the production system is in transition. Until recently, the directories used on the production system for most EPICS related software were rooted at /usr/local/pepii/. The primary directories in that file-tree will not be described here further. The file-tree for the planned directories, to which we have been recently moving, and whose structure is intended to mirror that found on Development, is rooted at /usr/local/cd/soft/. This new location therefore contains mainly host side software.

2.4. Differences between Development and Production directories

Note that the cvs/ and ref/ directories present on the development system are not present on production. That's because we don't need source code, nor the intermediate product of builds, on production. Instead, only the build products, such as executables are needed. The build products are in the release directories (dev/, new/, prod/).

tst/ does not exist on production because it is only used to contain symlinks to the build products directories under ref/ on development.

Directory	Important Subdirectories	Function
lib/		Archive (non-dynamic) libs (.a), these are not escalated.
{dev, new, prod }	 script/ solaris/lib/ solaris/bin/ solaris/pbin/ javalib/ disp/ matlab/ python/ ora/ include/	**The Release Directories** Platform independent non-executable scripts. Platform dependent object libraries. Platform dependent executable. Platform dependent executable of system software. java class and jar files Epics .dl files Matlab executable and .m files Python scripts Oracle files, SQL etc Include files shared among applications
ioc/	<epics-version>/@	IOC applications software. Symlink to ../ref/epics/<version>/

Table 5 Important top level directories on Production systems

2.5. Environment Variables for Subdirectories

The developers login environment defines many environment variables for navigating the directory structures on both Development and Production. Those pertaining to the primary directories described above, are listed in Table 6.

Table 6: Environment Variables for Primary Directories

DEVELOPMENT (AFS, eg flora		PRODUCTION (NFS, eg gateways)	
Physical Dir.	**Env. Variable**	**Physical Dir.**	**Env. Variable**
$CD_SOFT/cvs	CVSROOT	Not applicable	
$CD_SOFT/ref $CD_SOFT/tst	CD_REF CD_TST	Not applicable	
$CD_SOFT/dev	CD_DEV	$CD_SOFT/dev	CD_DEV
$CD_SOFT/new	CD_NEW	$CD_SOFT/new	CD_NEW
$CD_SOFT/prod	CD_PROD	$CD_SOFT/prod	CD_PROD
$CD_SOFT/bck	CD_BCK	Not applicable	

3. File Protections

This chapter describes the file protections (ACLs and permission bits) of the important directories of the unix development environment. The file protections concentrate on the development host. Files on the production hosts are protected only by being in the cddev account, to which few people have login access and so is not supposed to be an interactive login account, so it is not discussed further here.

The directory systems with specially designed file protections are:

1. The root of the unix development directory tree hierarchy at /afs/slac/g/cd/soft/ ($CD_SOFT) and below.
2. The CVS repository rooted at $CD_SOFT/cvs/
3. The Reference area of the files in CVS, and where we build software, at $CD_SOFT/ref/
4. The Release directories: $CD_SOFT/tst/, $Cd_SOFT/dev/, $CD_SOFT/new/ and $CD_SOFT/prod/

These are described further below:

3.1. Top Level Directories cd/ and cd/soft

The development system is located in the AFS filesystem of the tailored machines maintained by SCS, at /afs/slac/g/cd/soft/. It therefore uses mainly the AFS "Access Control List" (ACL) system for file protections (see AFS Users Guide). The ACLs for cd and cd/soft are summarized in Table 7 :

Table 7 Present ACLs for cd and cd/soft

Directory	ACL protection groups	Mode bits	Members
/afs/slac/g/cd/	g-cd	rlidwka	jjo, brooks, greg, luchini zelazny
/afs/slac/g/cd/soft/ (CD_SOFT)	owner-g-cd	rla	jjo, brooks greg,luchini
	g-cd	As above	As above
	g-cd:soft	rlidwka	Everyone

The owner organization of the principal ACL entries for these directories is given in Figure 1.

3.2. CVS and Reference Directories' File Protections

This section describes the file protections of directories maintained by the CVS version control system, and the "reference" directories for them. The CVS directories are all under $CD_SOFT/cvs/, the reference area at $CD_SOFT/ref/.

Table 8: CVS and REF area ACL Protection Groups

Directory	ACL Protection Group	Mode bits
/afs/slac/g/cd/soft/cvs (and all dirs below)	owner-g-cd:soft g-cd:soft g-cd	rla rlidwk rlidwka
/afs/slac/g/cd/soft/ref (and all *reference*[4] dirs below)	owner-g-cd:soft g-cd:soft g-cd	rla rlidwk rlidwka

To write to a CVS directory, or a reference directory, you must be a member of g-cd:soft.

The directories are protected from accidental writes or deletions by having their NIS user mode bit "w" turned off (see below). The combination of the "w" ACL mode bit being present in the g-cd:soft protection group entry on the directories, but the user "w" NIS bit being normally absent on the files in those directories, protects the files, as described below.

3.2.1. The AFS ACL/NIS bits Interface

From the SLAC AFS User's Guide: "If the w mode bit is present (in the user bits of the NIS permission bits), anyone with the write and lookup rights on the ACL of the file's parent directory can modify the file; if the w bit is off, no one can modify the file, not even the owner." *This is true even if the file is owned by a different user!* So, if a file in the repository is owned by zelazny, greg can write it if the NIS user permission bit for write is on (ie, -rw-r—r--), and not otherwise. Furthermore greg can chmod u+w the file, even though he is not the owner, as long as he is a member of an ACL entry of the directory that has the w mode bit present (eg rlid**w**k).

3.2.2. Changing Protections When Updating the CVS Repository

The CVS program itself turns the NIS "w" bit on for files in the repository itself ($CD_SOFT/cvs) when it wants to update the file, such as during a "cvs commit", or cvs import" operation.

3.2.3. Changing Protections When Updating the CVS Reference

We have special scripts which run "in the background" when a user performs a cvs commit operation which updates the repository. These scripts automatically run a "cvs update" on the corresponding reference directory (under $CD_SOFT/ref/). These are run using the CVS "post-commit" facility and change the NIS permission bits to allow them to write (see 3.2.1 above).

Given this model for making temporary permissions changes, the permissions for files in $CD_SOFT/ref/ are handled by:

1. The cvs commit precommit operation sets the NIS user permission bit of the repository file "on" (chmod +w) before doing a cvs commit.
2. The cvs commit postcommit operation sets the NIS user permission bit of the repository file "off" (chmod -w) after completing the cvs commit.

The scripts are $CD_SOFT/ref/common/tool/cvs-precommit and cvs-postcommit.

[4] Remember that directories under $CD_SOFT/ref/ are split into two kinds, "reference" directories (of CVS), and the first stage release directories. See 3.3, for protection groups for the release directories.

3.3. Release Directories' File Protections

The release directories, $CD_SOFT/tst/, $CD_SOFT/dev/, $CD_SOFT/new/ and $CD_SOFT/prod/, and below, are protected by the fact that normally, there are no developers who are members of any AFS protection group which has write privilege to them. However, there is one protection group, g-cd:soft-rel-write, which does have the "w" mode bit set. g-cd:soft-rel-write protects the /tst/, /dev/ and /new/ directories. So programmers can temporarily get write privilege to those dirs by adding themselves to that group. After they have finished with updating the directory, they remove themselves from the group.

Only members of g-cd:soft-rel-write's owning group, g-cd:soft-rel-lib, may add themselves to g-cd:soft-rel-write. Therefore, write access to the release directories is restricted to members of the software group and some others, by only putting those trusted users in g-cd:soft-rel-lib. An identical scheme works to ensure only system managers can write to the "new" delivery directories for system tools, using ACL groups g-cd:soft-sys-lib and g-cd:soft-sys-write.

3.3.1. addUserRefWrite and removeUserRefWrite

The scripts `$CD_REF/common/tool/addUserRefWrite` and `removeUserRefWrite` manage adding users to g-cd:soft-rel-write and removing them in a controlled way. These utilities are called by the release management script `gmakerel`, which performs release escalation, to allow the user temporary access to the release directories. The utilities may also be called directly should the need arise. addUserRefWrite and removeUserRefWrite also manipulate g-cd:soft-sys-lib and g-cd:soft-sys-write to ensure only system managers can release to *new*.

As a further protection for the "last-chance" production software, $CD_SOFT/prod/ is owned by the **cddev** account on development machines, and only that account may write to it. This is because only cddev is in group g-cd:soft-prod-write. cddev is also a permanent member of g-cd:soft-rel-write , so that the sweep can delete files from NEW.

Table 9 summarises the ACLs. Note that g-cd:soft and g-cd are not among the protection groups on the release directories. That is to prevent members of those groups having write access, which would bypass the protection through g-cd:soft-rel-write.

Table 9 ACL Protections of release directories

Directory	Important ACL Entries	Mode bits	Normal Members
$CD_SOFT/{ref[5],dev,new}/ script/.. disp/.. solaris/.. sys/.. (etc)	g-cd:soft-rel-lib g-cd:soft-rel-write	rla dwk	everyone no-one
$CD_SOFT/new/sys/..	g-cd:soft-sys-lib g-cd:soft-sys-write	rla dwk	sys-admin no-one
$CD_SOFT/prod/disp/.. solaris/.. (etc)	g-cd:soft-prot-write	rladwk	cddev

[5] The directories under $CD_SOFT/ref/ which are rlease directories (those shown) are symlinked from $CD_SOFT/tst/, so it is actually the ACL of the $CD_SOFT/ref/ directories on which the protection groups are placed.

3.4. The ACL Protection Group Hierarchy

There is a deliberate hierarchy to the ACL protection groups we use. A developer can only change properties of a protection group, such as to add themselves, if they are a member of the group's owning protection group. Membership of the groups is described in the tables above, the ownership hierarchy is the figure below.

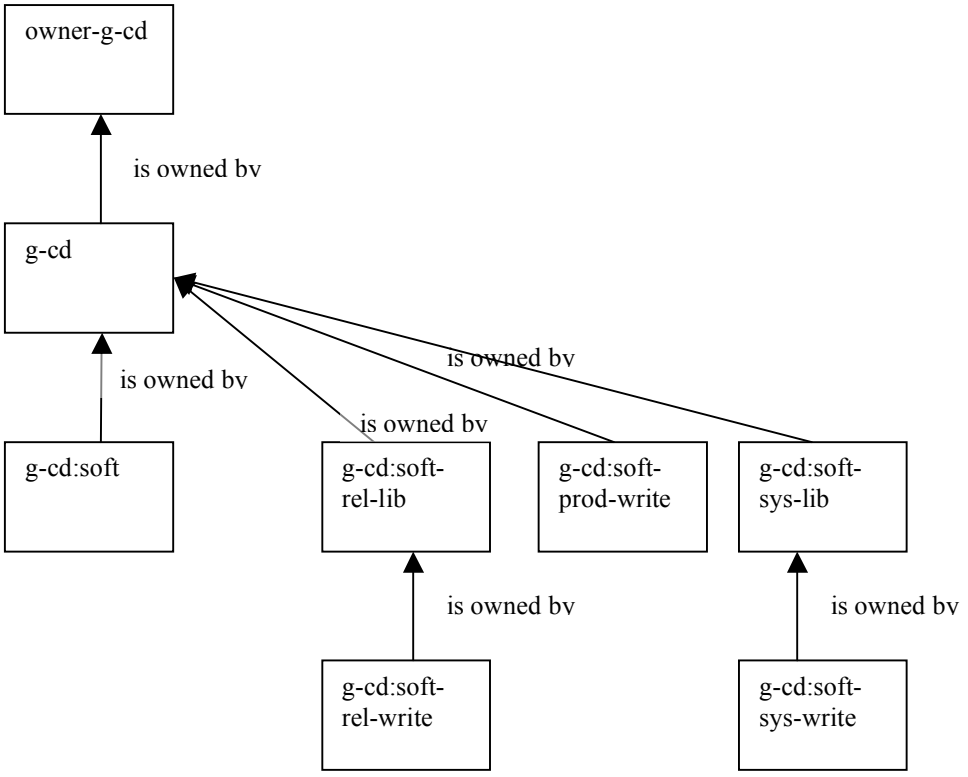


Figure 1 Protection Group Hierarchy

3.5. Actions for New Users

When new developers join the software group, we have to add them to the g-cd:soft and g-cd:soft-rel-lib protection groups. Someone in g-cd has to type a command of the form:

```
pts adduser –user <user> -group g-cd:soft
pts adduser –user <user> -group g-cd:soft-rel-lib
```

The user must klog before the pts adduser will take effect.

4. CVS and REF Directory Structural Layout

This chapter describes where the different kinds of software, such as EPICS display definitions, scripts, compilable source code etc, that are in CVS. Use this as a guide when creating new CVS modules in our repository.

This cannot be an exhaustive list since directories will be added frequently. We give here only a feeling of the intention of the top levels.

Table 10 CVS Major Directories

Top Level	Subdirs	Intended Use
$CD_SOFT/cvs/common/	.	Files not specific to a single application. Typically ASCII, defined by example, such as scripts, the makefiles defining the release procedure, etc.
	setup/	Scripts defining development environment.
	tool/	Script utilities
	make/	Makefiles defining build system
$CD_SOFT/cvs/gui/	.	Graphical display configuration files.
	disp/	EPICS display definitions (dm, dm2k etc).
$CD_SOFT/cvs/app/	.	Source code and/or configuration files for application level programs
	channelWatcher/	source and configs for channelWatcher program.
	alh/	source and configs for Alarm Handler program.
$CD_SOFT/cvs/util	.	Source code and/or configuration files for utilities and libraries (contrast with $CD_SOFT/cvs/app/).
	alh_hrtbeat_mon	Source code for alh heartbeat monitor.
$CD_SOFT/cvs/ioc/	.	IOC Applications source code and configs.
	R3.13.1	Code specific to R3.13.1
	R3.13.2	Code specific to R3.13.2
	R3.13.6	Code specific to R3.13.6

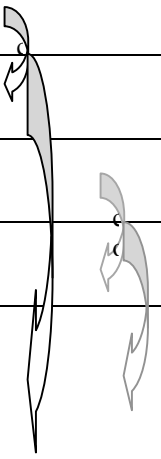
4.1. Reference Directory Structure

In general, for each cvs directory under $CD_SOFT/cvs/ (Table 10), there is a corresponding “reference directory” under $CD_SOFT/ref/. These contain a permanently checked out copy of the files in CVS. We use the reference directory files to actually build the control system. Additionally, they can be used “as reference”, so you don’t have to check out a cvs module or use a cvs browser like CVSWEB to look at the files in CVS.

However, the reference directories have an important additional role; this is where programs and displays are built. Since the intermediate build products, such as object files, are made in subdirectories of the source code directory by our makefiles, the reference directories additionally contain build products. Those are typically in subdirectories named O.<host-architecture>/, such as O.solaris/. Thirdly, the top level reference directory $CD_SOFT/ref/ also contains the directories which contain the final outputs of the builds, the executables themselves. See Table 11 for examples of what goes on under ref/.

Table 11 Example REF directories. The example shows 2 EPICS display directories containing display source code (what's in CVS), plus the intermediate build output directories (O.solaris), plus the final delivery directory (disp) which contains the output from builds of displays from all the EPICS display O.solaris directories.

Directories	Files
$CD_SOFT/ref/gui/disp/config/pepii/	pepii_tt_main.adl someotherdisp.adl
$CD_SOFT/ref/gui/disp/config/pepii/O.solaris/	pepii_tt_main.dl someotherdisp.dl
$CD_SOFT/ref/gui/disp/config/tarf/	tarf.adl makerfcool.adl
$CD_SOFT/ref/gui/disp/config/tarf/O.solaris/	tarf.dl makerfcool.dl
$CD_SOFT/ref/disp/	pepii_tt_main.dl someotherdisp.dl tarf.dl makerfcool.dl



5. The Software Build System Framework

This chapter describes the makefile framework which implements the building (compiling and linking), installation, and release, of Unix Control System Software.
The objective here is to describe those aspects of the build system which a programmer would have to know in order to add support for handling additional file types, or to modify handling of existing file types, rather than to describe the system's mechanism in detail.

Programs we develop ourselves for ourselves (as opposed to 3rd party packages we adapt, and packages we develop for possible outside consumption), use a variation of the EPICS makefile system we have developed, to define the build. That makefile framework can build a project for a number of target platforms, and is intended to provide a level of compiler and linker consistency across all our programs. Additionally, the makefiles manage software release through the release directories.

5.1. References

See the Basic Users Guide for how the system described here is used by developers.

For a description of the EPICS IOC applications makefile system, on which our makefile system is based, see the EPICS document IOC Software Configuration Management. Our system was based on the R3.13.2 version of IOC Configuration makefiles.

For the list of recognized makefile macros which a developer uses to specify a build, such as PROD, SRCS, USR_LDFLAGS and so on, see specifically the IOC Software Configuration Management part 4.3 Description of Makefiles[6],

5.2. The CD SOFT Makefile Framework and a Project's Makefiles

The makefile framework files (sometimes confusingly called "config" files), we have defined, are in `$CD_SOFT/ref/common/make/`. These are a specialization of the EPICS R3.13.6 makefiles, in `/afs/slac/package/epics/R3.13.6/base/config/`. That is, the EPICS makefile system includes a mechanism for customizing the makefiles for a given site, like ours, though the CONFIG_SITE* files. We have had to additionally change some other files, to support things like release escalation, so we have our own versions CONFIG_SITE files, plus a some others. See the references above for help with the EPICS makefile system, and the list of supported macros used to define builds in the system.

Each project must have a very short file named Makefile in each directory (whose basic job is just to list the subdirectories to be searched for source code to build) and it must have a Makefile.Host in each source code subdirectory, which defines what to build. If your program is simple, and you just have a single directory, so it contains the source code, then you'll have both these two makefiles in that directory:

> `Makefile`: This file is necessary, but you only need to change it from the template to add DIRS specifications for each source code subdirectory of your program. It define the architectures for which to build and the subdirectories to be searched for more Makefiles.
>
> `Makefile.Host`: This specifies the build.

You can find templates of these in `$CD_REF/common/stds/unix/template/`.

The idea is a program's directory only needs the two short makefiles in its directory to use the makefile framework. Those two makefiles, Makefile and Makefile.Host, include the makefiles of the framework. The Makefile.Host specifies *what* to build. The

[6] http://epics.aps.anl.gov/asd/controls/epics/EpicsDocumentation/AppDevManuals/iocScm-3.13.2/buildingComponents.html#DescriptionOfMakefiles

framework files define *how* to build the project's deliverables from it's sources, and release the deliverables.

For examples of how to write Makefile and Makefile.Host in order to build a project, see the relevant chapters of the Basic Users Guide. That document guides a programmer in developing displays, scripts, and programs, and so details what to put into Makefile and Makefile.Host in order that they properly interact with the framework files.

5.2.1. Required Structure of Makefile and Makefile.Host

Both Makefile and Makefile.Host must define a variable TOP, to point relatively to the "project root". In practical terms this is "enough `../` such that, when the project is in ref, TOP would be $CD_SOFT/ref/." E.g, the Makefile in `$CD_SOFT/ref/common` defines "`TOP = ..`"; and in `$CD_SOFT/ref/common/tool/Makefile`, "`TOP = ../..`". Note, though, that Makefile.Host is intended to be run from the host specific object directory that Makefile creates, which is an immediate subdirectory of the source directory, so the TOP= in Makefile.Host always has one more set of "`../`" than the TOP in Makefile, so in `$CD_SOFT/ref/common/tool/Makefile.Host`, "`TOP = ../../..`"

5.2.2. Makefile Framework Hierarchy

The framework proceeds hierarchically from Makefile and Makefile.Host:

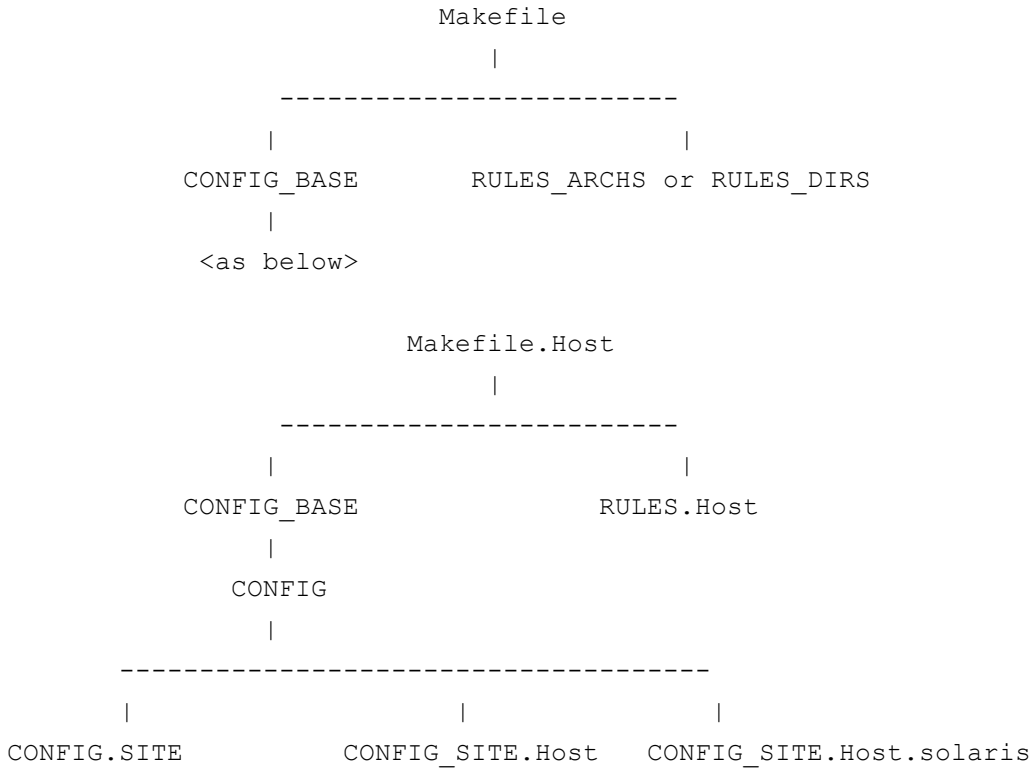


Figure 2: makefile hierarchy in $CD_SOFT/ref/common/make ($CD_COM_MAKE)

There are a few more makefiles in $CD_SOFT/ref/common/make: RULES_TOP, RULES_JAVA, RULES.Unix are not included by any file! But they are standard EPICS makefiles and can be included in Makefiles.

Makefile	Primary Uses
CONFIG_BASE	Defines INSTALL delivery directory roots (LIB, BIN, DISP ect), and default HOST type
CONFIG	Includes our own versions of files below.
CONFIG_SITE	Compiler and linker options; directory specifications, like defining INSTALL, DEV, NEW and PROD delivery directory specifications, macros for defining which files are at which level of release (eg NEW_PROD, INNEW_BINS).
CONFIG_SITE.Host	Shell command definitions: MV, RMDIR etc.
CONFIG_SITE.Host.solaris	Directory specifications for tools and libraries of utilities used in the Control System, eg Oracle, X11, gnu, Motif, Perl, Python. Solaris compiler switches and system include file directories (for –I specs), system library directories (for –L and -l specs) etc.
RULES.Host	The build rules; the targets "build" "install" "buildInstall" "dev" and "new", plus all the rules for installing what from where to where, moving to dev and moving to new.
RULES_ARCHS	What a build consists of (ACTIONS)

Table 12: The makefile framework "config" files and some of their primary uses. Those in BOLD are the important ones for defining how to handle each functional filetype's install and release.

The most important of the EPICS R3.13.6 makefiles a programmer needs to be aware of, is $EPICS_BASE/config/CONFIG_COMMON, which contains the compiler and linker options used for each compiler.

5.3. Makefile Support for Install and Release

This section lists the important sections of the makefile framework config files which are concerned with file install and release. Install and Release is primarily handled by CONFIG_SITE and RULES.Host.

5.3.1. CONFIG_SITE

Defines the macros for the determining the list of files of each functional filetype, which should go to the install directory:

```
BUILD_ARCHS              = $(HOST)

# Define "install" dirs.
#
# These define where files will be installed by the "install" target. Install
# target is that which moves the build products from where they are left by
# the "build" target (under a given project directory, such as O.solaris),
# to the directory common to all
# projects. The install target is the default target in fact - unless you
# specifically say "gmake build", the build will be completed and the results
# installed. In the ESD Software version of the EPICS build system, the
```

```
# "install" target is that which puts things into the "tst" stage of release
# (ie moves stuff to to /ref/<install-dir>), or, if developing in a working
# directory, just into <projectroot>/<install-dir>,
# like ~/myproj/sun4-solaris2/bin/).
#
INSTALL_LOCATION          = $(TOP)
INSTALL_LOCATION_LIB      = $(LIB)
INSTALL_LOCATION_BIN      = $(BIN)

INSTALL_BIN             = $(INSTALL_LOCATION_BIN)
INSTALL_LIB             = $(INSTALL_LOCATION_LIB)
INSTALL_SHRLIB          = $(INSTALL_LIB)
INSTALL_SCRPT           = $(INSTALL_LOCATION)/script
INSTALL_DISP            = $(INSTALL_LOCATION)/disp
INSTALL_dmDISP          = $(INSTALL_LOCATION)/dm-disp
# The following functional file type instal locations have
# not yet been implemented:
INSTALL_CONFIG          = $(INSTALL_LOCATION)/build
INSTALL_INCLUDE         = $(INSTALL_LOCATION)/include
INSTALL_ORA             = $(INSTALL_LOCATION)/ora
INSTALL_MAT             = $(INSTALL_LOCATION)/matlab
INSTALL_HTML            = $(WWW_DIR)
INSTALL_JAVA            = $(JAVALIB)

# "dev" target release directories.
#
# Note re Non-executable Scripts: There is no escalation of non-executable
# scripts past DEV, so there is no  NEW_SCRPT or PROD_SCRPT. This is
# because there is no search PATH function for scripts, so there is no
# way to implement an automatic release scheme.
#
DEV_LOCATION  = $(INSTALL_LOCATION)/../dev
DEV_BIN       = $(DEV_LOCATION)/$(BUILD_ARCHS)/bin
DEV_LIB       = $(DEV_LOCATION)/$(BUILD_ARCHS)/lib
DEV_SHRLIB    = $(DEV_LIB)
DEV_DISP      = $(DEV_LOCATION)/disp
DEV_dmDISP    = $(DEV_LOCATION)/dm-disp
DEV_SCRPT     = $(DEV_LOCATION)/script

# "new" target release directories.
#
NEW_LOCATION  = $(INSTALL_LOCATION)/../new
NEW_BIN       = $(NEW_LOCATION)/$(BUILD_ARCHS)/bin
NEW_LIB       = $(NEW_LOCATION)/$(BUILD_ARCHS)/lib
NEW_SHRLIB    = $(NEW_LIB)
NEW_DISP      = $(NEW_LOCATION)/disp
NEW_dmDISP    = $(NEW_LOCATION)/dm-disp

# "sweep" target release directories.
#
# "sweep" target is defined only in $CD_SOFT/ref/Makefile.
#
PROD_LOCATION = $(INSTALL_LOCATION)/../prod
PROD_BIN      = $(PROD_LOCATION)/$(BUILD_ARCHS)/bin
PROD_LIB      = $(PROD_LOCATION)/$(BUILD_ARCHS)/lib
PROD_DISP     = $(PROD_LOCATION)/disp
PROD_dmDISP   = $(PROD_LOCATION)/dm-disp

#===============================================================
#
# Map functional-file type to install directory.
#
# These macros pre-pend the install
# directory pathname to the head of each file defined in the functional
# filetype macro (for instance, prepends INSTALL_BIN to the head of each
# file in the PROD macro defined in Makefile.Host. The result will match,
# at runtime after macro expansion, a rule in the list of INSTALL_* rules
# toward the end of RULES.Host.
#
INSTALL_PROD          = $(PROD:%= $(INSTALL_BIN)/%)
INSTALL_LIBS          = $(LIBNAME:%= $(INSTALL_LIB)/%)
```

```
INSTALL_SHRLIBS       = $(SHRLIBNAME:%=$(INSTALL_SHRLIB)/%)
INSTALL_SCRIPTS       = $(SCRIPTS:%= $(INSTALL_BIN)/%)
INSTALL_SCRPTS        = $(SCRPTS:%= $(INSTALL_SCRPT)/%)
INSTALL_DISPS         = $(DISPS:%= $(INSTALL_DISP)/%)
INSTALL_dmDISPS       = $(dmDISPS:%= $(INSTALL_dmDISP)/%)
# The following are not supported yet:
INSTALL_HTMLS         = $(HTMLS:%= $(INSTALL_HTML)/$(HTMLS_DIR)/%)
INSTALL_CONFIGS       = $(CONFIGS:%= $(INSTALL_CONFIG)/%)
INSTALL_ORAS          = $(ORAS:%=$(INSTALL_ORA)/%)
INSTALL_MATS          = $(MATS:%= $(INSTALL_MAT)/%)
INSTALL_INC           = $(INC:%=$(INSTALL_INCLUDE)/%)
INSTALL_OSINCLUDE     = $(INSTALL_INCLUDE)/os/$(ARCH_CLASS)
INSTALL_OSINC         = $(OSINC:%= $(INSTALL_OSINCLUDE)/%)

# Map functional file type to DEV release directory.
#
# As defined above for the install directory, one DEV release directory
# must be defined for each functional file type. These have a slightly
# more compilated
# syntax than the INSTALL* macros above only because the directory part
# of the functional file-type list macros' values (<PROD>, <SCRIPT>) and
# so on, must be stripped off. They must be stripped because there will
# be a directory name in the value of
# PROD, SCRIPT etc, in the case of releasing software that is not
# in the same directory as the Makefile.Host file, such as releasing external
# packages from afs/slac/package through Makefile.Hosts under
# $CD_SOFT/ref/ext/.
#
DEV_PROD              = $(patsubst %,$(DEV_BIN)/%,$(notdir $(PROD)))
DEV_LIBS              = $(patsubst %,$(DEV_LIB)/%,$(notdir $(LIBNAME)))
DEV_SHRLIBS           = $(patsubst %,$(DEV_LIB)/%,$(notdir $(SHRLIBNAME)))
DEV_SCRIPTS           = $(patsubst %,$(DEV_BIN)/%,$(notdir $(SCRIPTS)))
DEV_SCRPTS            = $(patsubst %,$(DEV_SCRPT)/%,$(notdir $(SCRPTS)))
DEV_DISPS             = $(patsubst %,$(DEV_DISP)/%,$(notdir $(DISPS)))
DEV_dmDISPS           = $(patsubst %,$(DEV_dmDISP)/%,$(notdir $(dmDISPS)))

# Map functional file type to NEW release directory.
#
# As defined above for the install directory, one NEW release directory
# must be defined for each functional file type. These again have a slightly
# more compilated than for INSTALL or DEV above, because in this case
# the list of files of each functional file-type must be found by looking
# in the DEV release directory, and filtering out all those file that were
# not in the user's Makefile.Host's functional file-type assignment. For
# instance, to make NEW_PROD (the list of executables that should be escalated
# to new, we look at all the files in DEV_BIN (INDEV_BINS), and filter that
# list for those which were in the users PROD list (PROD), which are also
# presently at the dev level of release (DEV_PROD).
#
INDEV_BINS       = $(wildcard $(DEV_BIN)/*)
INDEV_LIBS       = $(wildcard $(DEV_LIB)/*)
INDEV_DISPS      = $(wildcard $(DEV_DISP)/*)
INDEV_dmDISPS    = $(wildcard $(DEV_dmDISP)/*)

NEW_PROD         = $(patsubst $(DEV_BIN)/%,$(NEW_BIN)/%,$(filter $(INDEV_BINS),
$(DEV_PROD)))
NEW_SCRIPTS      = $(patsubst $(DEV_BIN)/%,$(NEW_BIN)/%,$(filter $(INDEV_BINS),
$(DEV_SCRIPTS)))
NEW_LIBS         = $(patsubst $(DEV_LIB)/%,$(NEW_LIB)/%,$(filter $(INDEV_LIBS),
$(DEV_LIBS)))
NEW_SHRLIBS         = $(patsubst $(DEV_LIB)/%,$(NEW_LIB)/%,$(filter $(INDEV_LIBS),
$(DEV_SHRLIBS)))
NEW_DISPS        = $(patsubst $(DEV_DISP)/%,$(NEW_DISP)/%,$(filter $(INDEV_DISPS),
$(DEV_DISPS)))
NEW_dmDISPS      = $(patsubst $(DEV_dmDISP)/%,$(NEW_dmDISP)/%,$(filter
$(INDEV_dmDISPS), $(DEV_dmDISPS)))

# Define pre-requisite locations for sweep Makefile.
#
INNEW_BINS       = $(wildcard $(NEW_BIN)/*)
INNEW_LIBS       = $(wildcard $(NEW_LIB)/*)
```

```
INNEW_DISPS      = $(wildcard $(NEW_DISP)/*)
```

5.3.2. RULES.Host

RULES.Host specifes the build rules, that is, what to do to build, install, and release (to dev or new) each kind of file. Notice that the system knows how to install (to the first level of release

```
all::   install

build:: inc

build:: $(LIBTARGETS) $(dmDISPS) $(PROD) $(INSTALLS)
        @echo "\n"$(BOLD)"Actions to build successfully completed"
$(NOBOLD)

inc:: $(INSTALL_INC) $(INSTALL_OSINC)

rebuild:: clean install

install:: buildInstall
        @echo "\n"$(BOLD)"Actions to install successfully completed"
$(NOBOLD)

buildInstall :: build $(TARGETS) $(INSTALL_PROD) $(INSTALL_LIBS)
$(INSTALL_SHRLIBS) $(INSTALL_SCRIPTS) $(INSTALL_HTMLS) $(INSTALL_CONFIGS)
$(INSTALL_ORAS) $(INSTALL_SCRPTS) $(INSTALL_MATS) $(INSTALL_dmDISPS)
$(INSTALL_DISPS)

# Release escalation targets for dev and new levels of release.
#
# SCRPTS are not escalated past dev.
#
dev: $(DEV_PROD) $(DEV_LIBS) $(DEV_SHRLIBS) $(DEV_SCRIPTS) $(DEV_SCRPTS)
$(DEV_dmDISPS) $(DEV_DISPS)
        @echo "\n"$(BOLD)"Actions to release to DEV successfully
completed" $(NOBOLD)

new: $(NEW_PROD) $(NEW_LIBS) $(NEW_SHRLIBS) $(NEW_SCRIPTS) $(NEW_dmDISPS)
$(NEW_DISPS)
        @echo "\n"$(BOLD)"Actions to release to NEW successfully
completed" $(NOBOLD)
```

RULES.Host additionally contains all the INSTALL, DEV and NEW rules, that manage local installation, and moving to dev and new.

Note, the Left-Hand-Sides of these rules actually match the instantiations of the INSTALL_*S, DEV_*S and NEW_*S macros defined in CONFIG_COMMON. That is, those macros expand to fully qualified file names, which then match these Left-Hand-Sides:

```
$(INSTALL_BIN)/% : %
        @echo "\nInstalling executable $? to $(INSTALL_BIN)"
        @$(INSTALL_PRODUCT) -d -m 555 $? $(INSTALL_BIN)

$(INSTALL_LIB)/%.a: %.a
        @echo "\nInstalling library $? to $(@D)"
        @$(INSTALL_LIBRARY) -d -m 644 $? $(INSTALL_LIB)
…
$(DEV_BIN)/% : $(INSTALL_BIN)/%
        @$(INSTALL_DEV) -d -m 555 $? $(DEV_BIN)

$(DEV_LIB)/%.a: $(INSTALL_LIB)/%.a
        @$(INSTALL_DEV) -d -m 644 $? $(DEV_LIB)
…
```

```
$(NEW_BIN)/% : $(DEV_BIN)/%
        @$(INSTALL_NEW) -d -m 555 $? $(NEW_BIN)

$(NEW_LIB)/%.a: $(DEV_LIB)/%.a
        @$(INSTALL_NEW) -d -m 644 $? $(NEW_LIB)
```

5.4. Developing and Testing Framework Makefiles

A nice feature of our makefile system, is that the system for build, install, dev release, or new release, can all be developed and tested very easily from a developer's working directory. That is, even if working on the system for "gmake new", the system will do release escalation all locally in the developer's working directory, without affecting the real “new” release directories in /afs/slac/g/cd/soft/new.

No change to any makefile or environment variable is needed. Since the tree under $CD_SOFT/ref/ is just the meta-project of all projects, and that includes the release escalation system, and all operations are relative to TOP, exactly the same gmake commands work in your local directory as in $CD_SOFT/ref/ and they deliver to places relative to your working directory.

However, since the primary function of gamketst, gmakedev and gmakenew, are to unlock the file locking protections for the $CD_SOFT release directories and to log you actions, you have to use the pure gmake commands that each of those commands wrap, see Table 13: gmake wrappers for unlocking protections on release directories.

Command	Wraps
gmaketst	**gmake**
gmakedev	**gmake dev**
gmakenew	**gmake new**

Table 13: gmake wrappers for unlocking protections on release directories

So, to test modifications to the "dev" release system macros and such, type "gmake dev" instead of "gmakedev". Recall that "prod" release is done by the sweep, which is defined entirely in the sweep rule in $CD_SOFT/ref/Makefile.

5.4.1. Overriding which makefiles are used

If you have made local changes to common/make, and want to test them, override the definition of INCMK (don't use ~/work, ~ is only recognized by c-shell, but gmake does shell operations in bourne). E.g.:

c**d ~/work/app/project**

gmake dev INCMK=$HOME/work/common/make

The above simulates a release to dev in your working directory (in this case it will put build products under ~/work/dev/…) using local makefiles.

5.5. To Add Handling for a New File Type

This section describes how you would add to the makefile framework files to add a new “functional file-type”. The functional file-types are, for instance, SCRIPTS, DISPS, LIBRARY and so on. They are designations of a “kind” of file that the makefile system knows how to deal with. Each functional file-type goes into some directory, but more than one functional-file type can go into one directory. For instance, both PROD and SCRIPT functional file types go into the INSTALL_PROD directory. Sounds

complicated? It's not; we only need to change 2 makefiles to add support for a new functional file type, CONFIG_SITE and RULES.Host.

In the following example, we will add support for a new functional file type "CONF", which was added to be the way to release general kinds of configuration file: That is, we want the makefiles to understand a line like this in a Makefile.Host:

```
CONF = elogTemplate.xml
```

The following assumes you know how use CVS. Also, *CONF is only released to dev, so the following only describes how to support releasing a new kind of file up to dev*. However, support for releasing to new is also done in these same files – just follow the pattern; sweeping is done by $CD_SOFT/ref/Makefile, and you will have to edit that file.

5.5.1. Checkout common/make

The makefile system framework files are in common/make

```
cvs co common/make
```

5.5.2. CONFIG_SITE

We need to add macros for CONF to CONFIG_SITE. These are from various places towards the top of CONFIG_SITE. The existing lines are omitted for clarity:

```
INSTALL_CONF  = $(INSTALL_LOCATION)/conf
…
DEV_CONF      = $(DEV_LOCATION)/conf
…
INSTALL_CONFS = $(CONF:=%= $(INSTALL_CONF)/%)
…
DEV_CONFS     = $(patsubst %,$(DEV_CONF)/%,$(notdir $(CONF)))
…
```

5.5.3. RULES.Host

We will add to RULES.Host in two places: the first is to add to the prerequisites of the rules that say **what** to build when someone types "gmake", and the second is where it says **how** to build those things:

First, add to the buildInstall and dev rules:

```
buildInstall :: build $(TARGETS) \
    ...
    $(INSTALL_CONFS)

dev: $(DEV_PROD) $(DEV_LIBS) $(DEV_SHRLIBS) $(DEV_SCRIPTS)
$(DEV_SCRPTS) $(DEV_dmDISPS) $(DEV_DISPS) $(DEV_INCLUDES)
$(DEV_CONFS)
        @echo "\n"$(BOLD)"Actions to release to DEV successfully
completed" $(NOBOLD)
```

Then tell RULES.Host how to make INSTALL_CONFS and DEV_CONFS. Although it doesn't immediately look like the following does that, because the left-hand-sides don't seem to match the right-hand-sides of the rules above, they do, honestly! Once gmake has expanded INSTALL_CONFS, the resulting files match the left-hand side of the following rules:

```
$(INSTALL_CONF)/%: %
        @echo "\nInstalling config file $? to $(@D)"
        @$(INSTALL) -d -m 444 $? $(INSTALL_CONFIG)

$(INSTALL_CONF)/%: ../%
        @echo "\nInstalling configuration file $? to $(@D)"
        @$(INSTALL) -d -m 444 $? $(INSTALL_CONF)
```

Note, the apparent mismatch between INSTALL_CONF**S** and INSTALL_CONF. That isn't a mistake – check what we added to CONFIG_COMMON.

Now add the same rule for escalating CONFS to dev

```
$(DEV_CONF)/%: $(INSTALL_CONF)/%
        @$(INSTALL_DEV) -d -m 444 $? $(DEV_CONF)
```

Two rules were added for INSTALL_CONF and only one for DEV_CONF? That is because CONF files may be drawn from the directory in which Makefile.Host runs (eg O.solaris), or from the directory above – hence two rules. But once CONFs have been released to dev, they can only be in INSTALL_CONF ($CD_SOFT/ref/conf/).

5.5.4. Test supporting a new file-type

Test your additions in your working directory (see 5.4). Note that the makefile system will create the delivery directories for you (locally, if you do this in your own area it won't do anything in $CD_SOFT):

```
gmake INCMK=${HOME}/dev/conf/common/make
gmake dev INCMK=${HOME}/dev/conf/common/make
```

If your new filetype macro releases all the way to NEW, you must also teach the sweep how to sweep your files. Sweeps are done by the Makefile right at the root, $CD_SOFT/ref/Makefile. Be very careful when editing this file not to introduce unintended tabs at the beginning of the lines in the rules, since these are just very long shell commands executed by the makefile, and make treats tabs specially.

5.5.5. Release your makfile support for new file-type

If all is well, release the makefiles (CVS commit common/make).

Then do a release that uses the new functional file type you added (CONF in the above example). The release escalation system will create the delivery directories for you (in $CD_SOFT/ref/ and $CD_SOFT/dev/), just as it did when you tested, and put your CONF files in them.

The distribution system will also automatically create the delivery directories on production (on the gateway) if you put lines like `conf/elogTemplate.xml` in the manifest file of the directory you released.

5.5.6. Set $CD_SOFT/tst symlink and set Protections

For a new delivery directory root (ref/conf/, dev/conf, etc are "delivery" root directories), do 2 additional things:

1. Add symlink from $CD_SOFT/tst/ to $CD_SOFT/ref/ for the delivery dir, e.g.:

```
cd $CD_SOFT/tst
ln -s ../ref/conf conf
```

If you have created a new source code area, create the symlink to that from CD_SOFT/src/.

2. Set the protection groups on the ACL of the delivery directory for the new files type, in each of ref, dev, plus new and prod if necessary (you will need to be a member of protection group g-cd to do this) E.g.:

```
cd $CD_SOFT/ref/conf
fs setacl -dir . -acl g-cd:soft-rel-lib rla
fs setacl -dir . -acl g-cd:soft-rel-write dwki
fs setacl -dir . -acl g-cd:soft none
fs setacl -dir . -acl g-cd none

cd $CD_SOFT/dev/conf
fs copyacl -fromdir ../../ref/conf -todir .
fs setacl -dir . -acl g-cd:soft none
fs setacl -dir . -acl g-cd none
```

When you've finished it should look like this...

```
[slcs6]/afs/slac/g/cd/soft/dev/conf> fs listacl
Access list for . is
Normal rights:
  g-cd:soft-rel-write dwk
  g-cd:soft-rel-lib rla
  owner-g-cd:soft rla
  system:slac rl
  system:administrators rlidwka
  system:authuser rl
```

That's all. The above completes support for CONF.

5.6. Software Distribution System

This section describes the design and control flow of the remote distribution scripts used in the UNIX Development Environment release procedure.

The distribution scripts (distdev, distnew, distprod) are called by the gmakedev, new, prod, etc. after the gmake has completed. They are run from the /afs/slac/g/cd/soft/dev/script area.

All of these scripts are wrappers to a master forward distribution script, fwddist. This script also resides in /afs/slac/g/cd/soft/dev/script.

The fwddist script takes the following arguments:

- $1: Master directory. This is the root directory of your release escalation environment.
- $2: Source directory. The directory from which files will be distributed (relative to $1)

- $3: Comparison directory. The script will look in this directory for matching files to compare to files in the source directory (relative to $1)
- $4: Manifest file path.
- $5: Distfile name, path fully specified.
- $6: "sync" Distfile name, path fully specified, to be used for the synchronization phase.
- $7: YES to run the script in test mode. NO to do the actual distribution.

Some important points to note about the scripts:

The flow of control is pretty straightforward. The script is pretty well commented, so you should have no problem following it. The script:

- generates a list of files to distribute, based on manifest files

- creates a temporary Distfile containing these files to distribute

- calls rdist with the temporary Distfile as an argument

The release procedure is pretty much the same for all levels, except that for "prod", the script starts at the top of $CD_REF and goes through all manifest files, whereas for other release levels, it looks only in the current working directory and its subdirs for manifest files.

fwddist looks for a file called "manifest" in the release directory. If it does not find one, the script will exit normally; it will just think that it has no files to send.

The files listed in the manifest file assume that the root dir is the Master directory($1) specified by the command-line argument. Look at the arguments for the wrapper scripts(distdev, distnew, etc.), for an example of how fwddist is called.

The fwddist script depends on one external file called fwddist.dft,which is also in /afs/slac/g/cd/soft/dev/dist. This is what is called a Distfile, which is needed by rdist. However, it is really just a template distfile, which is read in by fwddist, which then adds release-specific parameters to the file and writes a temporary distfile to /tmp. Note that the /tmp Distfile name is created using the release escalation level(dev, new, prod) and a timestamp, so it is highly unlikely that two release Distfiles would ever have the same name at the same time.

The version of rdist that we use is NOT the same one that is supported by SCS. So, please use the following man page for an explanation of the command-line arguments:

http://www.magnicomp.com/rdist/7.0/doc/rdist.html

... the arguments we use ARE different from the UNIX man page on rdist, so, please use this one instead.

More info, albeit outdated, is also available at the following page:

http://www.slac.stanford.edu/grp/cd/soft/unix/dev/slaconly/RemoteDistribution.html

5.6.1. Distribution Directory Synchronization

For the “new” release escalation level, the fwddist script checks for the existence of the files it successfully copied to the “new” directory on production in the “dev” directory on production. To do this, it sends a remote shell command through ssh to check for the presence of the file, and then calls rm remotely to remove this file. Note that this can take some time.

For the “prod” release escalation level, fwddist moves all files contained in manifest files in the CD_SOFT reference area to the “prod” directory. To delete the files from “new”, the fwddist script does a directory synchronization

command, where it rewrites the “new” directory on production to mirror the “new” directory on development. This generally executes much faster than the iterative delete in the “new” release procedure.

6. Control System Host Process Management

This chapter describes the requirements, objectives, design and implementation, of the various mechanisms for managing host (eg opi, as opposed to ioc) level processes in the Unix parts of the SLAC Control System.
Such process are things like the so-called "command server" (cmdSrv), and the processes it itself manages (such as epics display requests), Channel Watchers, archiver engines, gateway processes, cmlog, AIDA processes, etc. At the time of writing a list of many of these processes, and the hosts on which they run, is available at http://www.slac.stanford.edu/grp/cd/soft/share/slaconly/network/opi/index.html.

6.1. Functional Requirements and Design Objectives

The following is a list of the functional and design objectives of the processes management system described in this chapter:

1. Programmer only has to setup the environment in one place to avoid the possibility of environment conflict or error.
2. The mechanism for starting, restarting, killing, or just showing the status of a process, is uniform across processes, and easy to use.
3. The user should not have to know the physical node on which the process usually runs, or the arguments for ports etc, the system should know that.
4. Support the fact that many, but not all processes, have an instance for each "accelerator" (pepii, pack, nlcdev), and additionally may have a development instance.
5. There is some mechanism for starting all the processes in a family together, such as all AIDA processes.
6. There is a single unified environment, though that environment is cleanly bifurcated by:
    a. Development (afs) and Production (nfs) nodes
    b. Development (compiling, building) and runtime (execution)
    c. Accelerator (PEPII, NLCDEV, PACK, default "PROD", DEV, etc).
7. The environment under which each process runs is defined for that process at runtime, so the processes environment is not subject to conflicts or errors made elsewhere, and processes started by cmdSrv can get environment updates without restarting cmdSrv and so restarting all cmdSrv processes on a host.
8. Developers can easily characterize any new processes they create, so they know which process management tools have to be changed in order to support that process.

6.2. Different Ways to Start a Process & which processes they start

Developers login	all = regular scripts, cs scripts, st type II scripts
cddev login on prod	all
cddev login on dev	all
unixfoyer	cs scripts
cmdSrv	cs scripts
SCP	cs scripts(via cmdSrv), ssh on VMS invokes script on prod unix

host	
watchdog	st type II
host startup	st type II
cron job	regular scripts; may or may not need environment
warmst	st type II
<process>manager	st type II

Table 14 Ways to start a process

The object is to unify the way in which processes are started, and the environments in which they are started.

6.3. Environment Definition

This section describes the shell script files which define the basic development and runtime environments for the control system.

6.3.1. The Basic Environment Definition (ENVS.csh)

A tcsh shell script, $CD_SOFT/dev/script/ENVS.csh, sets the basic environment for all logins and processes. That is, it includes all of the core development environment for developers, and it additionally defines the core runtime environment of the control system. That is, it sets up a significantly different environment when run on a development node to when it is run on a production node.

6.3.2. EPICS environment for each accelerator

Each “realm” or “accelerator” of the control system requires some specific EPICS setup. The environment for each realm is initialized by a csh shell script named epicsSetup<realm>. For instance epicsSetupDev, epicsSetupProd, epicsSetupPepii, epicsSetupNlcdev.

On development systems, ENVS.csh sets up the EPICS “Dev” environment using epicsSetupDev. On production systems it sets the EPICS “Prod” environment. Since the EPICS setup required on different accelerators (PEPII, NLCDEV, TARF etc) requires different EPICS values, users, cs scripts, and st files, should additionally source the appropriate EPICS Setup script (see templates below). It’s intended that ENVS.csh and the EPICS setup scripts are designed to allow multiple invocations, each overwriting the previous environment.

6.4. Environment Setup for each process

From the use cases given in Table 14, it appears that the following must each set up their own environment, in a similar way: 1) Cddev .cshrc (prod/dev), 2) Developer login, 3) Type II (st) scripts, and 4) cs* scripts. That is, 1) and 2) are login environments, and 3 and 4) are non-login. The Control System is mainly made up for these non-login processes.

6.4.1. Interactive Login Environments

This sub-section deals with how the environment is set up for type 1) and 2) – login, environments.

6.4.1.1. Environment setup for login to cddev

source /afs/slac/g/cd/soft/dev/script/ENVS.csh in .cshrc

On both Development (afs) and production (nfs) nodes, the environment will be set by a call to the standard environment script ENVS.csh. The environment so set up is different on Development and Production though, because ENVs.csh includes logic to set things up differently depending on which of these it's run on.

Note however, the environment so set up is not intended to be a complete environment for all cmdSrv processes (which run under cddev), as was formerly the case. Each cmdSrv process is now required to set its own environment as described below under Type II st files.

6.4.1.2. Developers login

Just like the cddev login, a developers login should include ENVS.csh. They may do this directly by editing their .cshrc, or using joining the HEPiX cd group. The HEPiX scripts for cd will call ENVS.csh

6.4.2. Non-login Process Environments

This sub-section deals with the environment definition for those control system processes which will be started by cs and st (type II) files, which is the great majority.

The non-login processes have the significant property that there is often one instance of the process for each "accelerator", or "realm". For instance there is an archive engine for PEPII, and another for NLCDEV, so the same executable runs on different hosts.

Also, sometimes, these processes fall into families of that should be managed together, for instance you want to start all AIDA processes in one go.

The first of these difficulties is dealt with by defining the host for each process, for each accelerator, in ENVS.csh. The second is dealt with by a system of scripts used for starting, killing, restarting, families of processes.

Below is described first how each process is formally associated with the hosts on which it runs, for each "realm" for which it should run, and subsequently how the cs and st files that start processes know which EPICS environment should be set up for the process.

6.4.2.1. Process Host Definition for non-interactive processes

The name and hosts of each process is defined in setEnv.csh, which is called by ENVS.csh. The following is an extract from the relevant part of setEnv.csh. Note the process names and host names listed for each processes. See the comments for the syntax of process and host names.

```
### Processes.
# Process are each defined by their NAME and HOST, plus optionally the PORT
# on whoch they run.
# <process>_NAME is the name you type to warmst or other scripts
# <process>_HOST[_{DEV,PROD,NLCDEV,PEPII}] are the hosts on which <process>
#    runs. As many may be defined as necessary. Note that the _<accelerator>
#    is optional, you don't need to define it if there is only one host
#    on which the process runs; egs, ERRCLIENT and OOC_COSEVENT_SERVICE.
# <process>_PORT specifies a port, if necessary.
#
# CMDSRV
```

```
setenv CMDSRV_NAME               cmdSrv
setenv CMDSRV_HOST_DEV           opi00dev00
setenv CMDSRV_HOST_RF            opi00rfs00
setenv CMDSRV_HOST_PROD0         opi00gtw00
setenv CMDSRV_HOST_PROD1         opi00gtw01
setenv CMDSRV_HOST_PROD2         opi00gtw02
setenv CMDSRV_HOST_NLCDEV        opi00gtw04
setenv CMDSRV_HOST_PEPII         slcs1

# Orbacus Event service
setenv OOC_COSEVENT_SERVICE_NAME oocCosEventService
setenv OOC_COSEVENT_SERVICE_HOST opi00dev00
setenv OOC_COSEVENT_SERVICE_PORT 4546

# Error Client
setenv ERRCLIENT_NAME            errClient
setenv ERRCLIENT_HOST            opi00dev00

# AIDA
setenv AIDA_NAMESERVER_NAME      DaNameServer
setenv AIDA_NAMESERVER_HOST_DEV  slcs6
setenv AIDA_NAMESERVER_HOST_PROD opi00dev00

setenv AIDA_DASERVER_NAME        DaServer
setenv AIDA_DASERVER_HOST_DEV    slcs6
setenv AIDA_DASERVER_HOST_PROD   opi00dev00
```

Figure 3: Extract of setEnv.csh, showing part of that part of setEnv in which each process' name, and the hosts on which it runs, are defined. For some processes a port number is also defined.

The mechanisms for starting non-interactive st and cs scripts refer to the names and hosts then symbolically, rather than absolutely, so changing the host on which a processes for any accelerator, is done through the setEnv.csh file. For instance, watchdog, unixfoyer, warmst, procmanager and the processes family managers, refer to hosts by these names.

6.4.2.2. How does a process know which accelerator it's part of?

Each process has to define it's own environment, and that environment will be different from accelerator to accelerator (PEPII, NLCDEV etc), so how will processes initialization set up the correct environment for a processes that's used on many accelerators?

The answer is that the process' st or cs script compares the name of the actual host on which it has been started with each <process-name>_HOST_<accelerator> environment variable for the process, and determines for itself which environment to set.

See the example cs and st files below.

6.4.3. cs scripts and st files

cs scripts are those called by cmdSrv to start interactive processes like EPICS displays for a given accelerator[7]. Type II st files are those which start non-interactive "server" type processes. cs scripts are executable SCRIPTS (delivered to CD_SOFT/{tst,dev,new,prod}/sun4-solaris2/bin/. Type II st files are also executable scripts, though they are delivered with the STARTUP macro to CD_SOFT/{tst,dev,new,prod}/sun4-solaris2/sys/. The different delivery directory allows us to control, through AFS permissions, who may release them.

[7] Mostly! Some processes started by cmdSrv are non-interactive and transient, like the report generators, eg tr00_report.

Both cs and st files follow the same basic template. They first source ENVS.csh. Then, depending on the host on which they're being run, they source on or more of the epicsSetup<accelerator> files. Finally they can make some additional environment changes or additions before starting their executable:

CS script template:

```
source afs/slac/g/cd/soft/dev/script/ENVS.csh
source epicsSetup<accelerator> based upon host discovery
[source or inline <program-specific-overrides of the above>]
exec <executable-name>
```

Type II st script template:

```
source afs/slac/g/cd/soft/dev/script/ENVS.csh (or
       /usr/local/g/cd/soft/dev/script/ENVs.csh)
[source epicsSetup<accelerator>] based upon host discovery
[source or inline <program-specific-overrides of the above>]
<executable-name>
```

where executable-name = binary or script

For example, this is the st.cmdSrv file – the type II st file for the cmdSrv processes:

```
[tersk05]:app/cmdSrv/script> more st.cmdSrv
#!/bin/tcsh -f
#
# This Script executes (starts) the the commmand server (cmdSrv) for
# all accelerator modes.
#
# Auth:
#   3/16/04 Ron MacKenzie
#=====================================================================
# Mod:
#=====================================================================

#
# Turn off 'other write' priv on files
#
umask 002

# Source the basic environment setup.
#
if(-e /afs/slac/g/cd/soft/dev/script/ENVS.csh) then
  source /afs/slac/g/cd/soft/dev/script/ENVS.csh
endif

#
# Make additions to the basic environment depending on the hostmode (aka
# "accelerator") on which this invocation of cmdSrv is being made.
#

setenv hostname_ `hostname`

if ( $hostname_ == $CMDSRV_HOST_DEV ) then
  if (-e /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupDev) then
     source /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupDev
  endif
else if ( $hostname_ == $CMDSRV_HOST_RF || \
          $hostname_ == $CMDSRV_HOST_PROD0 || \
          $hostname_ == $CMDSRV_HOST_PROD1 || \
          $hostname_ == $CMDSRV_HOST_PROD2 ) then
  if (-e /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupProd) then
       source /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupProd
  endif
else if ( $hostname_ == $CMDSRV_HOST_NLCDEV ) then
  if (-e /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupNlcta) then
       source /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupNlcta
  endif
else if ( $hostname_ == $CMDSRV_HOST_PEPII ) then
  if (-e /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupPepii) then
```

```
        source /afs/slac/g/pepii/ctrl/prod/bin/solaris/epicsSetupPepii
  endif
endif

set path=($path /bin /usr/bin /usr/ucb /usr/local/bin /usr/openwin/bin
/usr/dt/bin /usr/local/X11R5/bin)
#
#   cmdSrv uses this to find it's control files (allowable host and cmd
files)
#   They are unique per host because /usr/ is a shared file system.
#
setenv CMD_FILES /afs/slac.stanford.edu/g/cd/soft/dev/confsys/`hostname`
#
#   Used by this script file to locate the binary executable image.
#
setenv CMD_BIN   /afs/slac.stanford.edu/g/cd/soft/prod/solaris/bin
#
#   write log file here.  Also unique per host gateway.
#
setenv CMD_LOG  /nfs/slac/g/cd/cmdSrv/`hostname`
#
#   close and open new log file this often
#   (file contains error/debug messages).
#
setenv CMD_LOG_NHRS 8

#   Start the server.  Send internal debug  output to file.
#   Append to old one if there was a log file from previous startup.
#
    if (-d ${CMD_LOG}) then
      ${CMD_BIN}/cmdSrv > /dev/null &
    else
      echo "cmdSrv startup failed.  Cant access database directory"
    endif
# end of file
```

6.5. Process Management Support

A framework of scripts is available to help manage host processes. These help you see what's running, start, restart, or kill processes in the control system, in a controlled way. In particular, they control the formal names of processes and the hosts on which they run. This is particularly useful when some executable has to run on specific hosts as part of the control system for a particular accelerator realm, like PEPII, NLCDEV, or RF, and when a software suite s made up of a large number of executables which all have to work together, like Aida.

6.5.1. Prerequisites for process management scripts

The pre-requisites of managing processes using this framework are that the processes formal name and hosts are defined in setEnv.csh (see Figure 3 above), and it is started by the execution of a type II startup file (st.* file in $CD_SOFT/{new,prod}/sun4-solaris2/sys/. Users must also be authenticated to cddev on each host on which they intended to start or stop processes.

The system is made up of the following scripts:

warmst <process-name> {show, start, restart, kill}

The idea is warmst knows how to start all host processes that can be started by type II startup files. warmst can show the present status of a given process, by name, or start, restart or kill it. For start and restart operations, you give the process name and warmst

knows the correct type II st file to run it, on the host on which warmst is run, and in the username which runs warmst. It's basically warmst's job to map process names to st files.

procmanager {all, <process-name> }{show, start, restart, kill} [<accelerator-realm>]

Since warmst must be run on the host on which you want to start a processes (by running its type II startup file), procmanager works out what host its argument process should be run on, and uses ssh to run an appropriate warmst command on that host to start the process. procmanager works out which host is the right one, by looking in the environment variable whose name is the same as the environment variable which defined the processes name, except it replaces the _NAME part of the environment variable with _HOST. Eg, for the command server process itself, setEnv.csh defines the following:

```
setenv CMDSRV_NAME        cmdSrv
setenv CMDSRV_HOST_DEV    opi00dev00
```

so procmanager knows that it should start cmdSrv on opi00dev00 when the realm argument is dev. procmanager runs warmst in the cddev account, so you must be authenticated to cddev to use procmanager (see BUG for help on cddev authentication).

aidamanager {all, <processname>} {show, start, restart, kill} [<accelerator-realm>]

Each program suite can optionally define a manager script which wraps procmanager. Its job is to know all of the processes just in that suite. In this example aidamanager contains a list of all the processes in the Aida suite, and can therefore, for instance, restart all and only Aida processes on dev with the single command:

aidamanager all restart dev

There are presently only two such process suite manager scripts, aidamanager and errmanager, but others are envisaged, like cmlogmanager.

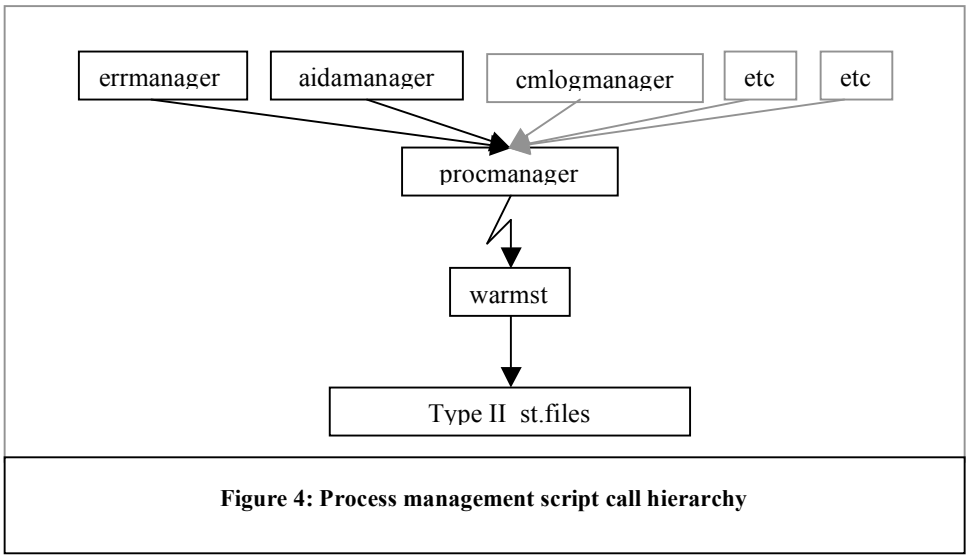


Figure 4: Process management script call hierarchy

7. CVS for IOC Software

7.1. Introduction

Some of our IOC software is different depending on which version of EPICS under which it is being run. Therefore, for some IOC source files we keep a different file "branch" for each version of EPICS in which it varies, and our CVS repository uses CVS "tags" to keep track of which branch of the file goes with which version of EPICS. In this way, we avoid keeping one repository module - which is a directory and all its subdirectories - for each version of EPICS. Instead, we keep only one module, called "ioc", and each file in it is tagged with which versions of EPICS it goes with. The versions of EPICS a file may go with are one or more of R3.13.1, R3.13.2 and R3.13.6.

7.2. CVS Repository and Reference Areas

Repository: `/afs/slac/g/cd/soft/cvs/`

Repository directory for IOC software: `/afs/slac/g/cd/soft/cvs/ioc/`
Reference areas:

```
/afs/slac/g/cd/soft/ref/epics/R3.13.1/ioc
/afs/slac/g/cd/soft/ref/epics/R3.13.2/ioc
/afs/slac/g/cd/soft/ref/epics/R3.13.6/ioc
```

There is only one CVS repository directory, or CVS "module" used for all three versions of EPICS for which we keep software. It's in /afs/slac/g/cd/soft/cvs/ioc/. There is no subdivision in that directory for each version of EPICS as there is in the reference area. Rather each individual file is tagged according to which versions of EPICS it belongs to, and a corresponding branch is created for each file that must be different for different versions of EPICS. The branch is given a tag name which names the version of EPICS to which that branch of the file belongs. For instance, for the file ioc/config/CONFIG we keep two branches, one for EPICS R3.13.6 and one for R3.13.1. If you check out CONFIG, you could check out the revision of that file at the head of either branch, and when you put it back into CVS, your new file would be the new head of only that branch. Now, every file in CVS must have a MAIN branch, this is, if you like, the "default" branch. All our R3.13.6 ioc software is on the MAIN branch. Therefore, we have not created a named branch for the R3.13.6 version of each file, it's just assumed that that is what's on the MAIN branch. So, only if a file specifically must be different for R3.13.1 or R3.13.2 we create a separate branch for that file. If we do create such a branch, it will be named BR3_13_1 for the R3.13.1 version, or BR3_13_2 for the R3.13.2 version. See Branching in the CVS Manual.

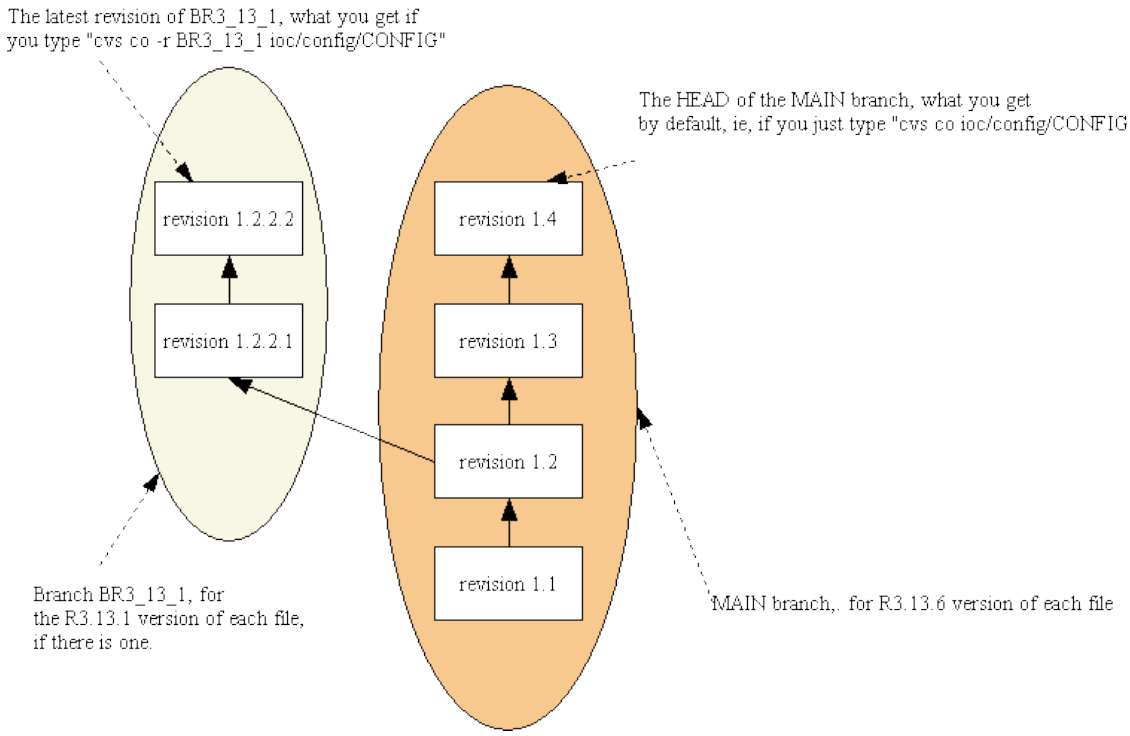


Figure 5 For ioc/config/CONFIG then, there are two branches, the MAIN branch, and BR3_13_1

7.3. The Reference Area

Unlike the repository, in the reference area we DO keep separate directories for each version of EPICS. The reference area is in /afs/slac/g/cd/soft/ref/epics/. That directory contains subdirectories for each of EPICS R3.13.1, R3.13.2 and R3.13.6:

```
[tersk08]/afs/slac/g/cd/soft/ref/epics> ls -l
total 20
drwxr-xr-x 3 greg  cd         2048 Jan 22 13:01 R3.13.1
        HEAD of MAIN branch unless there is a BR3_13_1 branch,
in which case the latest revision in that branch
drwxr-xr-x 3 greg  cd         2048 Jan 22 13:04 R3.13.2
        HEAD of MAIN branch unless there is a BR3_13_2 branch,
in which case the latest revision in that branch
drwxr-xr-x 3 greg  cd         2048 Jan 22 13:06 R3.13.6
        HEAD of MAIN branch
drwxrwxr-x 9 luchini        cd        2048 Jan 15 13:45 iocBoot
drwxr-xr-x 3 luchini        cd        2048 Mar 3  18:51 script
```

Figure 6: Example Reference area for a given file, ioc/config/CONFIG

For file ioc/config/CONFIG there is just the MAIN branch (which is always for the R3.13.6 version), plus one additional branch BR3_13_1 for the R3.13.1 version of CONFIG. There is no additional branch for R3.13.2, the version of CONFIG used for R3.13.2 is the R3.13.6 version - which as always is the one on the MAIN branch. Under each of the reference subdirectories for ioc/ there is reference copy of every file in cvs/ioc/, so all 3 of the reference subdirectories has an ioc/ subdirectory, and each of those contain a config/ subdirectory and within that subdirectory, a CONFIG file. So, for ioc/config/CONFIG, the two branches go into the three reference areas like this:

[tersk08]/afs/slac/g/cd/soft/ref/epics> find . -name CONFIG -print

./R3.13.1/ioc/config/CONFIG - The latest revision on the BR3_13_1 branch

./R3.13.6/ioc/config/CONFIG - The latest revision on the MAIN branch

./R3.13.2/ioc/config/CONFIG - The latest revision on the **MAIN** branch

This is accomplished using variations of the cvs checkout command, that use the -r and -f qualifiers, as described below.

7.4. To check out a file or directory

Use cvs checkout with the -r <tag> and -f options. -r <tag> says, "Get me the head revision of this file on the file's branch that is named <tag> if there is one", the -f says "but if there isn't any such branch for that file, then just get me the head on the main branch for that file." Note that -f should really only be used if you check out a directory, since if you check out a single file you should really know for which version of EPICS you want to check out that file.

7.4.1. To check out and modify more than one file in a directory

To checkout all R3.13.6 files in a directory, just issue a regular, unqualified, cvs checkout command, since R3.13.6 files are the MAIN branch. The following will check out the ioc/config:

```
cvs co ioc/config
```

To checkout the R3.13.1 versions of files in a directory in preference to the R3.13.6 versions, use branch BR3_13_1 with the -r and -f qualifiers. The -r BR3_13_1 says "get the version of the files tagged BR3_13_1 if there is one." The -f says "but if there is no version of each file specifically tagged BR3_13_1, then just get me the MAIN version (that one used for R3.13.6)." Eg:

```
cvs co -r BR3_13_1 -f ioc/config
```

To check out the R3.13.2 versions of files in a directory, in preference to the R3.13.6 versions, use branch BR3_13_2 with the -r and -f qualifiers. The -r BR3_13_1 says "get the version of the files tagged BR3_13_2 if there is one." The -f says "but if there is no version of each file specifically tagged BR3_13_2, then just get me the MAIN version (that one used for R3.13.6)." Eg:

```
cvs co -r BR3_13_2 -f  ioc/config
```

7.4.2. CVS commit directory of files

When you have completed your modifications, perform a cvs commit from your working directory. For instance, to commit everything you checked-out, cd up to your directory containing the ioc directory, and issue:

```
cvs commit -m "nice meaningful comment"
```

Luckily, the cvs commit command will take care of cvs updating all the relevant reference areas! This is accomplished by scripts we wrote, which are run in the background by cvs commit.

7.5. To check out and modify a single file in a directory

The difference for a single file is that you presumably know which version of EPICS your modifications pertain to, so you shouldn't be using the -f.

The following 3 examples of a cvs checkout command show how to checkout and modify the file ioc/config/RELEASE. That file exists in 3 different versions, one for R3.13.6, one for R3.13.2 and one for R3.13.1. The distinction is done through branches. The R3.13.6 version is on the MAIN branch, the R3.13.2 version is on a branch labeled BR3_13_2, and the R3.13.1 version is on a branch labeled BR3_13_2. See the actual CVSWEB entry for ioc/config/RELEASE to see this more clearly.

To checkout R3.13.6 files, just issue a regular, unqualified, cvs checkout command, since R3.13.6 files are the MAIN branch. The following will check out the ioc/config/RELEASE for R3.13.6. Eg:

```
cvs co ioc/config/RELEASE
```

To checkout the R3.13.1 version of a file in preference to the R3.13.6 version, use branch BR3_13_1 with the -r qualifier. The -r BR3_13_1 says "get the version of this file (or files, if the argument was a directory) tagged BR3_13_1. Eg:

```
cvs co -r BR3_13_1 ioc/config/RELEASE
```

To check out the R3.13.2 version of a file in preference to the R3.13.6 version, use branch BR3_13_2 with the -r qualifier. The -r BR3_13_2 says "get the version of this file (or files, if the argument was a directory) tagged BR3_13_2." Eg

```
cvs co -r BR3_13_2 ioc/config/RELEASE
```

7.5.1. CVS commit a single file

When you have completed your modifications to the checked out file, perform a cvs commit from your working directory:

```
cvs commit –m “meaningful comment”
```

The cvs commit will look after updating all the CVS reference areas for each version of EPICS in which the file you committed had a branch!

7.6. To create a branch for a file

If you want to make it so a file is different depending on which version of EPICS it's being used with, branch it. To do this you need to use a "branching tag" like "BR3_13_1". When these operations are complete, you can use cvs checkout commands like those above to get the specific branch you want. **Note that this is always only used for changing old revisions of a file, not the most recent revision** (you can't after all create a branch where there is no trunk); so the example here shows how to get the original file from R3_13_1, modify it, and put it back creating a new branch right off that first version of the file.

First checkout the file into your working directory:

```
cvs co -r R3_13_1 -f ioc/config/RELEASE
cd ioc/config
```

Then create the branching tag (cvs tag -b) for the file you checked out, giving the branch name. *Note, this operation takes effect on the repository straight away - no cvs commit need be issued!* The example below gives it the tag "BR_13_1". For branching tags there is a convention in CVS to use a tag name beginning with "B", because it's very difficult to make sense of tags and branches in CVS status and history listings (like CVSWEB) without a clue as to which tags were used for branching. For examples, see the table Tags used in our CVS repository for IOC Software below.

```
cvs tag -b BR3_13_1
```

Additionally, you should put the tag on your local copy of the file. Since the cvs tag command works on the repository, not on your working copy, you need to tell your working copy to use the new tag (not strictly necessary if you intend to cvs commit straight way without making any changes to the file, but who knows, you might get dragged into another project before you get to commit, in which case your changes to this file would then go to the head revision, not to the new branch)! To do that, cvs update:

```
cvs update -r BR3_13_1
```

You can then safely edit this file. Finally, CVS commit

```
cvs commit -m "Create branch for EPICS 3.13.1"
```

Our version of cvs commt will take care of updating each of the reference areas appropriately, assuming you have followed the branch naming convention strictly[8].

7.7. Tags naming convention in CVS repository for IOC Software

Note that the tag naming convention must be followed strictly in order for the cvs commit scripts to correctly update all the reference areas for each branched file.

Specifically, for files which are branched so that they can have a different instance for each versio of EPICS (or more than one version anyway), the branch name must begin “B”, and be followed by a string which names a directory under $CD_REF/epics/.

Tag name	Purpose
R3_13_1	The original version developed for R3.13.1
	There is no R3_13_2 tag used, since all the original, pre CVS, software we developed for R3.13.2 runs under R3.13.6, so we have not created a separate original tag for that software.
R3_13_6	The original version of a file developed for R3.13.6
BR3_13_1	The tag used for branches of a file created specifically to support EPICS R3.13.1
BR3_13_2	The tag used for branches of a file created specifically to support EPICS R3.13.2
BR3_13_6	The tag used for branches of a file created specifically to support EPICS R3.13.6

Figure 7 Tag Names for Branching IOC files under each version of EPICS

[8] Specifically, DO NOT USE "cvs update **-dA**" if for some reason you try to update the reference area by hand. The -dA instructs cvs to reset the sticky tags and get the HEAD of the MAIN branch, that's precisely what you don't want. That would have no effect on the R3.13.6 branch files, since they are the MAIN branch, but it would not be what you want when updating the R3.13.1 or R3.13.2 files.