ANGLO-AUSTRALIAN OBSERVATORY AAO/DRAMA_ORG_2
DRAMA Software Report 2
Version 0.2

Tony Farrell
29-Jul-1993

DRAMA Software Organisation

Contents

1 Introduction 2

2 Overview of Requirements 2

3 Overview of Basic Proposals 2

4 Systems and Sub-systems 3

5 Directory Structures 3
5.1 Release Directories 4
5.2 Version contents 6
5.3 Quick Fix Directories 6

6 Source Directories 7

7 Reverting to Previous Versions 7
7.1 Finding files 9

8 Special files 9
8.1 Operating System boot time file 9
8.2 Operating System shutdown file 10
8.3 DRAMA Start Files 10

A Sccs Groups and Classes 11
A.1 Details 12

B The VxWorks Dramastart Command 14

C Version Management Tools 17

1 Introduction

DRAMA is a modular software system which allows new instrumentation systems to built from modules which are *plugged* together. This provides a similar interface to various instrumentation configurations.

A system consisting of several sub-systems is much more complicated than the sum of its parts. This is because the overall system includes all the interfaces between the parts as well as the parts itself. It is clear that with increase in complexity, techniques are needed to manage that complexity.

This note discusses the problems that which have been encountered and proposes standards for the organisation of directories and for the management of software releases.

This note is not yet complete.

2 Overview of Requirements

Historical events have made us realise the importance of guaranteeing that-

- the release of a piece of software should be as the result of an explicit decision on the part of the programmer.
- software should always be tested as completely as possible prior to releasing it *and* after releasing it.
- it should always be possible to revert to previous working versions of software.

All these points may seem painfully obvious, but nevertheless problems have been occurred that could be attributed to the failure to observe one or other of them. It cannot be over-emphasised that a single seemingly innocent change which apparently achieves the desired effect can still cause a fatal error when a system is used in a slightly different way.

3 Overview of Basic Proposals

The purpose of this section is to give an idea of what we are trying to achieve. The following sections describe the standard in considerably greater detail.

The most important aspects of the standard are that

- It should be possible to run released instrumentation software from any appropriately authorised username.
- It should be possible to have multiple independent systems each using potentially different versions of the same sub-systems.
- It should be possible, by issuing a single command, to select or alter which version of a given system should be used.

- It should be easy to make quick fixes to running systems and to give control over whether such fixes are used.
- Every file required by a system should be categorised as belonging to one or other sub-system.
- All released instrumentation software should be held in release directories which are completely separate from development and pre-release directories.
- Prior to release, all modified instrumentation software should be kept in development and pre-release directories which are completely separate from source directories.
- All source should be kept in source directories which are completely separate from programmers' directories.
- All development should be performed in programmers' directories.
- It should be possible for programmers to test their modifications whilst these modifications are still in their own private directories, whilst accessing all other files from development, pre-release or release directories.

4 Systems and Sub-systems

The term "system" will be used to refer to the set of sub-systems that corresponds to an observing configuration. The are many examples, *e.g.* UCLES plus IPCS, UCLES plus CCD, RGO Spectrograph plus IPCS, RGO Spectrograph plus CCD, Taurus plus IPCS.

Such systems will be treated in most respects like ordinary sub-systems in that they will have their own names and directories. Each will have a few files that are specific to it, such as a c-task to control its sub-systems in synchrony and optionally to maintain a status display, DCL and ICL procedures to control startup and close down, and system-specific data assessment programs.

5 Directory Structures

On **VMS** machines, **DRAMA** is kept in this structure is DRAMADISK:[...]. There are two major sub-directories. DRAMADISK:[RELEASE] contains the **DRAMA** release itself in directories such as DRAMADISK:[RELEASE.DITS] and DRAMADISK:[RELEASE.SDS]. The directory DRAMADISK:[LOCAL] is optional and can be used to store local software which is to be part of the **DRAMA** version management structure.

On **Unix** machines, **DRAMA** is normally kept in a dummy account named "`drama`". This allows it to be accessed by with `~drama` (if the tilde format is supported by your shell). But on some machines it may be in another location - talk to the person responsible for setting up **DRAMA** on you machine. `~drama` is only needed once, you simply replace it by its equivalent if **DRAMA** is kept elsewhere. Within this structure, `~drama/release` contains the **DRAMA** release itself while `~drama/local` is the optional area for local software.

Note that the **DRAMA** directory structures on **VMS** and **Unix** machines are functionally equivalent. If you have disks which are NFS shared between **VMS** and **Unix** machines, they can have just one copy of the **DRAMA** directory structure.

5.1 Release Directories

It is important to realise that a released version of a system has to consist of everything. Of course, individual sub-systems will be released separately, but the only version that can be trusted to work is the complete set of all sub-systems as it existed at a certain time and even then it can only be trusted if all possible modes of operation were tested at that time and if none of the hardware or external software has changed! The re-release of *any* part of that complete working system can theoretically stop the entire system from working.

Released versions of systems and sub-systems will normally be held in DRAMADISK:[local.*subsystem.version*] under **VMS** and `~drama/local/`*subsystem*`/`*version* under **Unix**[1] For now, assume that all the files corresponding to a given version are lumped together in the same directory. In Section 7 we will discuss the desirability or otherwise of this approach. The version name is normally a character string of the form "`rn_m`", which stands for "Release n.m".

VMS

As a concrete illustration, assume that the CCDECH (CCD + Echelle) system consists of the CCD and ECHELLE sub-systems. Further assume that CCDECH is at Version 1.3, which corresponds to Version 1.4 of the CCD sub-system and Version 2.1 of the ECHELLE sub-system. The DRAMADISK: directory structure will therefore contain (at least)

```
DRAMADISK:[LOCAL]
    [.CCDECH]
        files to do with CCD and ECHELLE operating together
        [.R1_3]
            Released version 1.3 of CCDECH. Common Files.
            [.VAXVMS]
                Vax/Vms specific executable and object files etc.
    [.CCD]
        files to do with CCD operating alone
        [.R1_4]
            Released version 1.4 of CCD. Common files.
            [.VAXVMS]
                Vax/Vms specific executable and object files etc.
    [.ECHELLE]
        files to do with ECHELLE operating alone
        [.R2_1]
            Released version 2.1 of ECHELLE. Common files
            [.VAXVMS]
                Vax/Vms specific executable and object files etc.
```

[1]For the actual **DRAMA** sub-systems release, replace "local" by "release".

UNIX

On a **Unix** system, it could look like-

- `~drama/local`
  - `/ccdech`
    - files to do with CCD and ECHELLE operating together
    - `/r1_3`
      - Released version 1.3 of CCDECH. Common Files.
      - `/sun4`
        - Sun 4 specific executable and object files etc.
      - `/decstation`
        - Decstation specific executable and object files etc.
  - `/ccd`
    - files to do with CCD operating alone
    - `/r1_4`
      - Released version 1.4 of CCD. Common files.
      - `/sun4`
        - Sun 4 specific executable and object files etc.
      - `/decstation`
        - Decstation specific executable and object files etc.
  - `/echelle`
    - files to do with ECHELLE operating alone
    - `/r2_1`
      - `/sun4`
        - Sun 4 specific executable and object files etc.
      - `/decstation`
        - Decstation specific executable and object files etc.

You can see, that if **VMS** filenames are mapped to lower case **Unix** filename, then these two structures can be put together on one NFS shared disk, keeping software for all architectures on one machine.

The basic structure in either case is

- *Drama Location*
  - *subsystem*
    - The sub system.
    - *version*
      - Files common to all machine architectures.
      - *machine architecture 1*
        - Files for machine architectures 1.
      - *machine architecture 2*
        - Files for machine architectures 1.
      - *machine architecture ...*
        - Files for machine architectures ....

For this to work well, the names for the various architecture releases must be consistent. The following are currently defined-

- **vaxvms** A **VAX** running **VMS**.
- **sun4** A Sun 4 (Sparc station) running SUN OS 4.1 (Solaris 1).
- **sun4_solaris** A Sun 4 (Sparc station) running Solaris 2.
- **decstation** A Decstation (MIPS 3000 in little endian mode) running Ultrix.
- **vw68k** A 680x0 running the VxWorks real time operating system.

5.2 Version contents

What should a released version consist of?

All the operating system files current at the time of the release?	This is just not practicable, although CD rom has some possibilities here.
All the standard **DRAMA** system files current at the time of the release?	This is practicable. The total number of **DRAMA** files required at run-time and for program development is not too great and if they are regarded as being part of the release then we can be confident that existing versions will continue to run across new **DRAMA** releases (which otherwise would be major events).
Any other system used by Instrumentation Software, such as Starlink?	This may or may not be partical. Sub-systems should be aware of which other system there are using and define a sub-system specially for these files.
All the local sub-system files?	Obviously these must be present!

5.3 Quick Fix Directories

Mistakes are bound to occur. There are bound to be situations where a piece of software that is needed in half an hour refuses to work and it is not helpful to force the use of the release system in order to fix such a problem.

If a quick fix is necessary it *must not* be performed in the release directories (what if the quick fix introduces a worse bug than the one that it fixes—this is quite likely to happen?). Instead they should be performed in quick fix directories, which will have the same names as release directories except that the final part of the name will be “qn_m” (“q” for “Quick Fix”) rather than “rn_m”. Under **VMS**, These directories can be inserted into the search paths for all of the affected sub-systems. Under **Unix**, you will need to make a copy of the release directory into the quick fix directory, and then copy in the changed files.

It must be the responsibility of the fixer to report the problem to the programmer responsible for the sub-system. This will eventually result in the problem being resolved by the relevant programmer and the proper resolution finding its way via the source directory, the development directory and the pre-release directory to the release directory. In the mean time users will have to continue to use the quick fix (and programmers will have to be aware that the quick fix version of the sub-system in question is being used).

6 Source Directories

Source code should be managed by an appropriate version system, CMS on **VMS** machines and SCCS on **Unix** machines with SCCS much preferred due to its portability and the potential disappearance of **VMS** over time. (does anyone have a **VMS** version of SCCS?).

As mentioned in section 2, it is desirable that source be kept in directories separate from the programmers directories, to isolate the source from dependance on a particular programmer. Probably the best location is DRAMADISK:[LOCAL.*subsystem*.SOURCE] on **VMS** and `~drama/local/`*subsystem*`/source` on **Unix** are the best locations.

Also in section 2, it mentions the desirability of of development work being done in the programmers directories.

Under **Unix** the best approach is for the programmer to `acmmRcsCo DramaDrama` those files which make up a release into his directory. By then making a soft link (using `ln -s`) to the SCCS directory, the programmer can edit and replace SCCS files at will.

Under **VMS**, assuming the use of CMS, the best approach to probably to create a CMS reference directory. You then define a logical name search path which first points to your working directory and then the CMS reference directory. You then set your CMS directory the appropriate location and your default directory to the logical name.

If you wish to develop on the opposite machine type from the source library type (in order to make software run on both machines) then the best approach is probably to have the discs from your **VMS** machine NFS mounted on your **Unix** machine or vis-a-versa. You probably also need a multi-session terminal (such as an X-terminal or workstation). By keeping a session open on both machines and using a combination of the above techniques, you can normally develop a reasonable working arrangement. If you don't have these facilities, you will have to shuffle data between machines.

SCCS does not support a CLASS and GROUP system as **VMS** does. Regardless, it is desirable to be able to extract a given version at will and to be able to define a release group. Appendix A examines techniques which can be used for doing this based on using a *Makefile.* This relies on the the automatic fetching of *Makefiles* from SCCS directories. Note, that it is *not* recommended that you use this *Makefile* to actually build your system. Document [1] describes the *dmakefile* system, which can be used to build *Makefiles* for any supported system.

7 Reverting to Previous Versions

It has already been proposed that released software should reside in directories called `/drama-/local/`*subsystem*`/version` and `DRAMADISK:[LOCAL.`*subsystem*`.version]`. At any one time there will be several of these directories for each sub-system, *e.g.*

`DRAMADISK:[LOCAL.CCDECH.R1_2]`	Released version 1.2
`DRAMADISK:[LOCAL.CCDECH.R1_2.VAXVMS]`	Released version 1.2, **VMS** executables
`DRAMADISK:[LOCAL.CCDECH.R1_3]`	Released version 1.3
`DRAMADISK:[LOCAL.CCDECH.R1_3.VAXVMS]`	Released version 1.3 **VMS** executables
`DRAMADISK:[LOCAL.CCDECH.R1_4]`	Released version 1.4
`DRAMADISK:[LOCAL.CCDECH.R1_4.VAXVMS]`	Released version 1.4 **VMS** executables

Under **VMS**, all access to a sub-system's files at run-time will be via the job logical name *subsystem*_DIR. This is defined as a search path to firstly the common directory (say `.CCDECH.R1_2]`) and then the **VMS** executables directory (in this case `.CCDECH.R1_2.VAXVMS]`).

Under **Unix**, things are a bit more complicated since you cannot have search paths. The following environment variables must be defined-

*subsystem*_**DIR** points to the release directory structure.

*subsystem*_**DEV** points to utility programs etc. required for program development and suitable for running on the current host.

*subsystem*_**LIB** points to libraries required to build programs and libraries for the current target and programs built for the target.

It may not be clear at first why this level of complication is needed. The most common situation is developing software designed to run on the machine you are running on. In this case, *subsystem*`_DEV` and *subsystem*`_LIB` would point to the same directory.

However, other possibilities exist. When you use a Sun to develop software for a VxWorks machines, you need libraries appropriate to the VxWorks system, but compilers etc. appropriate to a sun. In this case, *subsystem*`_LIB` would point to the location of libraries built for use on the VxWorks system, while *subsystem*`_DEV` would point to the location of software appropriate to the sun.

Note that at some stage, to allow cross development with a **VMS** host, we will probably introduce a *subsystem*`_DEV` logical name.

When the appropriate startup command is done, (`DRAMASTART` under **VMS** and `~drama/dramastart` under **Unix**, `dramastart` under VxWorks), the appropriate logical names/environment variables are defined for the latest release of each system. This is appropriate for a software development environment but not Observer accounts.

The `DRAMA_VERSION` command provides a way of setting up logical names/environment variables for a particular version of an observing system.

```
$ DRAMA_VERSION 2AUG
```

would define the appropriate logical names for a version created on 2nd August. For the CCDECH system mentioned previously, it would define the CCDECH_DIR, CCD_DIR and ECHELLE_DIR logical names. See appendix C for details.

Version files are simply command files which define the logical names/environment variables and are created whenever a new version is released to observers.

7.1 Finding files

Once you have the logical names/environment variables, you can use them to find the various **DRAMA** files in a consistent. For example, if you want to link against the *dits* library, use

```
DITS_DIR:DITS.OLB
```

on **VMS** and

```
$DITS_DIR/libdits.a
```

on **Unix**. This will ensure you will always find the correct version.

When developing on a VxWorks based machine, things are a bit more difficult since environment variables [2] are not as easy to access from the **VxWorks** shell. Appendix B describes functions which provide an appropriate interface to this system.

8 Special files

Some files which may reside in the top level directory of a sub-system have special uses. The files normally refer to the default release of the sub-system.

8.1 Operating System boot time file

Various things are best done when the operating system boots. Under **VMS**, some programs will be installed with privileges and some shareable images will be installed so that they are indeed shareable. Under both **VMS** and **Unix**, daemons may need to be started. Under VxWorks, objects may have to be loaded.

Every sub-system which wishes to do one of the above should provide an OS boot time file. Under **VMS**, these are named *subsystem*_VMSSTART.COM. On a sun4, its *subsystem*_sun4start while under VxWorks, its *subsystem*_vw68kstart.

It can be assumed that all required files are in the latest release directories. Under **Unix**, the following environment variables can be assumed to be defined when this file is executed-

DRAMA Top level **DRAMA** directory.

DRAMA_HOSTTYPE Type of the current host (sun4, decstation).

DRAMA_TARGET Type of the target (sun4, decstation, vw68k).

They should be used to ensure startup files are location independent.

Under **Unix**, the file is executed using a bourne source command and should be written appropriately.

Details for **VxWorks** have not yet been determined.

[2] Introduced from **VxWorks** version 5.1

8.2 Operating System shutdown file

These files are similar to the above but are named *subsystem*_VMSSTOP.COM etc. They reverse what was done by the OS boot time files. They should be executed as part of an orderly OS shutdown.

Note that the *subsystem*_VMSSTOP.COM files have to be executed before changing the values of logical names that refer to installed images or de-installing or re-installing such images, and the *subsystem*_VMSSTART.COM files have to be executed afterwards.

8.3 DRAMA Start Files

At some time the *subsystem*_DIR logical name/environment variable must be defined. Once this is done, all reference to the sub-system should be via these logical names/environment variables. By doing this, you ensure that commands are independent of the actual version being used. Under **VMS**, the DRAMASTART command does this.

The DRAMASTART command will execute all the files named DRAMADISK:[LOCAL.*subsystem*]*subsystem*_DRAMASTART.COM. The main job of this file is to define the logical name *subsystem*_DIR to point to the latest release of *subsystem*.

Under **Unix** the command is-

```
~drama/dramastart [target] [command]
```

assuming there is an account named `drama` which the software resides in.

Target is optional. When not supplied, the target defaults to the host type. Currently, target is normally only specified when working on VxWorks software or if the second argument is given. In the first case, you specify a "`vw68k`". In the second case, you specify one of `sun4` etc. followed by a **DRAMA** command to be executed.

Dramastart will examine all the files named `~drama/local/`*subsystem*`/`*subsystem*`_dramastart-.rel`. These files are very simple, just containing a line `RELEASE=`*rel_num* where *rel_num* is the release number. The appropriate environment variables (*subsystem*_DIR, *subsystem*_DEV and *subsystem*_LIB) will be defined. Then either a new interactive shell is created or the *command* will be executed. If the user is a member of the group `drama`, the command `newgrp drama` is used to create the new shell.

Under **Unix**, to define the actual **DRAMA** commands, you should source the file `$DRAMA/drama.sh` if you use a bourne style shell or `$DRAMA/drama.csh` if you use a csh style shell. This may be done from your shell startup script, if there is one (as in csh's .cshrc file). The files `$DRAMA/local/drama.sh` or `$DRAMA/local/drama.csh` will be executed where and this is the location for local function definitions (sh) or aliases (csh).

Note that the **Unix** version of the `dramastart` command is more efficient if the environment variable `HOST` is defined to the host name and `HOSTTYPE` is define to the host type (sun4, decstation etc). It these are not defined, then commands must be executed to work them out.

Details for **VxWorks** are more complex and are described in appendix B. The same `.rel` files as the **Unix** software are used.

A Sccs Groups and Classes

When you execute the **Unix** *make* command in a directory containing no *Makefile*, but a *SCCS* directory, then that SCCS library is searched for the *Makefile*. If it exists, then a `acmmRcsCo DramaDrama` command is preformed for it and make run using it.

Under **DRAMA**, you would not uses this *Makefile* to build your software, as this is done using a *dmakefile* (see [1]). It is though quite acceptable to use this *Makefile* to assist in management of your SCCS directory. It need not be portable, accepts to machines to which you wish to move your SCCS directory.

Below is an example of such a makefile. This can be found in `~drama/demos/git_makefile`. This makefile implements a group called release and a class management system.

```
REL_DIR=.
DMKMF=/home/aaossc/drama/release/config/dmkmf

RELEASE = Git.h Git_Err.msg Sst.h dmakefile git.c git_dramastart.com  \
gitarggetd.c gitarggeti.c gitarggetl.c gitarggets.c \
gitargnp.c gitenvgets.c gitparenvgets.c gitpathget.c gitsimulation.c \
gittpi.c sst.c sst_main.c

.SCCS_GET:
        acmmRcsCo DramaDrama -s $@ -G$(REL_DIR)/$@


release : $(RELEASE)

vaxvms_release : $(RELEASE)
        (cd $(REL_DIR) ; $(DMKMF) -t vaxvms -f)

sun4_release : $(RELEASE)
        (cd $(REL_DIR) ; $(DMKMF) -t sun4)

decstation_release : $(RELEASE)
        (cd $(REL_DIR) ; $(DMKMF) -t decstation)

vw68k_release : $(RELEASE)
        (cd $(REL_DIR) ; $(DMKMF) -t vw68k)


newclass :
        @ if [ "$(CLASS)" = "" ]; then echo No class name supplied; exit 1;fi
        @ if [ "$(REMARK)" = "" ]; then echo No remark supplied; exit 1; fi
        @ if [ -f SCCS/s.class-$(CLASS) ]; then echo  class $(CLASS) \
  already exists ; exit 1 ; fi
        @ echo "#Class $(CLASS) created by $$USER at "‘date‘ >class-$(CLASS)
        @ echo "#$(REMARK) " >> class-$(CLASS)
        @ echo "# " >> class-$(CLASS)
        sccs prs -d"acmmRcsCo DramaDrama :M: -r:I:" $(RELEASE) >> class-$(CLASS)
        acmmRcsCo DramaDrama -e -p dmakefile |  sed 's/RELEASE=.*/RELEASE=$(CLASS)/' \
  >dmakefile
        sccs delta dmakefile -y"$(REMARK)"
        sccs create class-$(CLASS)
        sccs clean
        rm -f ,class-$(CLASS)
```

To fetch the group named `RELEASE` in to the directory `~/mydevdir`, you would do-

```
make REL_DIR=~/mydevdir
```

To create to new version, named `r0_1`, do

```
make newclass CLASS=r0_1 REMARK="put your release comment here"
```

If you later have to recover the source files as per release `r0_1`, you can do

```
acmmRcsCo DramaDrama class-r0_1
/bin/sh class-r0_1
```

The class creation details and your creation remark will be in the first couple of lines of the file `class-r0_1`.

As you can see, this file implements most of the important features of the **VMS** CMS group and class systems.

A.1 Details

We shall now work though this file.

The line `REL_DIR=../release` sets a default value for the location files are released to when you do *make release*. The default location is up to you, but it is often overridden on the command line.

The line `DMKMF=` defines the the *dmkmf* command. Replace `/home/aaossc/drama` by the relevant location, normally accessed by `~drama`[3]. Again, this command may be overridden on the command line.

The line `RELEASE=` defines the group `RELEASE`. Simply construct a make macro defining the files which make up the group.

The two lines introduced by `.SCCS_GET:` redefine the command executed when *make* attempts to fetch a file from SCCS. It causes the file to be put in the location defined by the `REL_DIR` macro.

[3]You cannot used ~drama, it does not work in *Makefiles*.

Next we have the various targets. The target `release` causes the files which make up the group `RELEASE`, as defined by the *make* macro of the same name, to be fetched into the release directory.

The following targets (`vaxvms_release` through `vw68k_release` will also release the files and in addition will run the *dmkmf* command to build a makefile for the particular target. These are convenience targets only.

The target `newclass` is the most complex target. You must specify values for the `CLASS` and `REMARK` macros on the command line and the first two lines of the implementation ensure this. The class name must be new (ensured by the line 3). It then creates a bourne shell script (lines 4-7) which will fetch the current generation of each file in the group defined by the macro `RELEASE`. This script is put in the SCCS directory (line 10). In addition, it will edit the file *dmakefile*, replacing the line `RELEASE=` by one setting the new class number (lines 8,9). It then removes junk files (lines 11,12).

To delete a class, delete the SCCS file created for the class. In the example above, delete `SCCS/s.class-r0_1`.

There is one major caveat to this system. If the release directory is different from the default directory (containing the SCCS directory), there must be no files belonging to the release group present in the default directory, either being edited or read. You should use *sccs clean* and *sccs info* to ensure this. If you don't, then those files will not be fetched.

B The VxWorks Dramastart Command

VxWorks machines are generally embedded systems and as a result, will in general have appropriate versions of all required software in ROM or it will be loaded from an appropiate place on the network at boot time. The major exception when such software is being developed.

When developing software targeted at **VxWorks** machines, the developer will normally have at least one interactive session running on the host **Unix** machine and one running on the target **VxWorks** machine. The later session will run an interactive session of the **VxWorks** shell. See [2] and [3] for details. We assume here that the **VxWorks** machine has NFS mounted the development directories on the **Unix** machine and that you are running **VxWorks** version 5.1 or higher.

On the host **Unix** machine, the programmer would have done a command of the form-

```
~drama/dramstart vw68k
```

assuming the target machine is a 680x0 machine. This will allow the progammer to build software for the specified target (vw68k). If the programmer executes the command "`echo $DRAMA`" on the **Unix** machine, he will be able to determine the actual location of **DRAMA**.

You now need to determine how to access this directory from the **VxWorks** machine. As mentioned above, we assume it is NFS mounted on your **VxWorks** machine. You can determine the mount point with the **VxWorks** function `nfsDevShow`. For example if "`echo $DRAMA`" on the **Unix** host `aaossc` produces-

```
/home/aaossc/drama
```

and `nfsDevShow` in the **VxWorks** target shell produces

```
device name         file system
-----------         -----------
/usr                aaossc:/usr
/usr/local          aaossc:/usr/local
/home               aaossc:/home
/var/spool/mail     aaossc:/var/spool/mail
value = 0 = 0x0
```

then you can see the `/home` on host `aaossc` is mounted at `/home` on the target. In this case, **Unix** paths starting with `/home` will have the same name on the **VxWorks** target. Things are more difficult if the mount point is different. For example If `/home/drama` is the location of drama on the host and it is mounted at `/usr/drama` on the target, then you should replace the prefix `/home` in **Unix** pathnames with `/usr` to get the **VxWorks** path.

Now assuming the above example, you can examine the contents of the directory with the **VxWorks** command "`ls /home/aaossc/drama`". This might produce-

```
.
..
drama_make
drama.csh
drama_source
drama_make.com
release
dramastart
demos
local
drama.sh
drama.vw68k
value = 0 = 0x0
```

The file we are interested in at this stage is `drama.vw68k`. This file is a **VxWorks** 680x0 object which implements the various functions described below. It is the first thing to load. Load it with a command such as

```
ld </home/aaossc/drama/drama.vw68k
```

With this file loaded, the **VxWorks** version of the `dramastart` is available. Execute it by typing `dramastart` at the **VxWorks** shell prompt.

VxWorks `dramastart` will examine all the *subsystem*/*subsystem*`_dramastart.rel` files and create appropriate *subsystem*`_LIB` and *subsystem*`_DIR` environment variables in a similar way to the **Unix** `dramastart` command. In addition, the environement variables `DRAMA` `DRAMA_TARGET` and `DRAMA_HOST` are created. You should now use these enviroment variables to access the **DRAMA** files, which ensures such references are independent of the software releases involved.

Unfortunately, there is one problem here. The **VxWorks** shell does not understand references to environment variables. For example

```
ls "$DITS_DIR"
```

does not work. To get around this problem, the `drama.vw68k` module implements replacements for the most common **VxWorks** shell commands which are affected. For example, `ls` is replaced by `dls` and the following will work as expected.

```
dls "$DITS_DIR"
```

as does

```
dls "${DITS_DIR}"
```

If the name of an enviroment variable is not surrounded by braces, then the name is terminated by the first unescaped, non alphanumeric character, except that in this case underscores and minus signs are considered alphanumeric. You can escape any character using a backslash character. If no value can be found for the variable, then it is replaced by an empty string. The commands provided are

dputenv Defines a new environment variable.

dlog Enables/disables logging in these functions.

dld Load a module. (replaces `ld`).

dls List contents of a directory. (replaces `ls`).

dll Long listing of a directory. (replaces `ll`).

dcd Change default directory. (replaces `cd`).

drun Load a module and run a program in it.

A full description of these commands now follow. Note that the **VxWorks** function `envShow` can be used to get a list of all the environment variables.

C Version Management Tools

To be written.

References

[1] Tony Farrell, AAO. *12-Feb-1992, Creating Makefiles for* **DRAMA** *Programs.* Anglo-Australian Observatory **DRAMA** Software Document.

[2] Wind River Systems **VxWorks** *Release 5.1 Programmer's Guide.*

[3] Wind River Systems **VxWorks** *Release 5.1 Reference Manual.*