RHIC/AP/3
August, 1993
T. Satogata, C. Saltmarsh, S. Peggs

RAPIDE 0.0

RHIC Accelerator Physics Intrepid Development Environment

Table of Contents

Section	Title	Page number
0	Introduction	2
1	Basic Environment Shells and groups Master .login and .cshrc files Windowing Text editors Documentation Backups and archiving	3
2	Controls environment Shared project directories Makefiles RCS (Revision Control System) LAMBDA directories ISTK (Integrated Scientific Tool Kit) Holy_Lattice Sybase database access Deliberate omissions	8
3	General environment Compilers and libraries Plotting and postscript Symbolic algebra CCD software Usenet news Sybase Metafont, TeX, LaTeX and dvi Liken Lattice design and tracking tools	18

0 Introduction

This document is a guide to the common environmental features of computing in (and around) the RHIC Accelerator Physics section, on the 'zoo' cluster of UNIX workstations, in RAPIDE, the RHIC Accelerator Physics Intrepid Development Environment. It is hoped that later revisions of this document will approach a more professional 'style guide', beyond the convenient collection of pointers and hints presented here.

RAP does two kinds of computing, "controls" and "general", addressed in sections 2 and 3 of this document. For general computing, efficient system administration requires cooperation in using a common environment. There is a much stronger need to define - and adhere to - a commonly agreed set of styles (or rules) in developing controls software. Right now, these rules have been set "de facto". Future improvements to the controls environment, particularly in response to the opinions of users, depends on broad knowledge of what the rules are.

There are environmental issues that are basic to both controls and general computing, and that are so fundamental that they are (almost) unarguable. They are described immediately below, in the next section.

1 Basic environment

Shells and groups

Users on the zoo domain almost exclusively use csh (C shell) or tcsh, an enhanced version of csh. Changing a user's login shell requires system administration privileges. csh is the default shell for new users as well as development environments (such as executable scripts). Because there are many common places where codes are shared, such as the LAMBDA area, all RAP users have a default group "accphys" and a default umask which gives read/write access to both the owner and group. In this way, code created and moved by one user is still available to another for modification. The recommended way in which editing collisions (two users editing the same file) are prevented is described in the section on RCS, the Revision Control System.

Master .login and .cshrc files

A set of standard .cshrc and .login files are present on the zoo domain as:

```
/home/owl/public/master_cshrc
/home/owl/public/master_login
```

These files cover setup of all the standard software available on the zoo domain machines. This is not meant to infringe on the individuality of separate resource files; instead it provides a solid base to every user, designed with the specifics of the zoo domain in mind. Use of these scripts allows administrative changes, such as installation of new software and changes in old, to be implemented with minimal strain on users. It is *strongly* suggested that all users use these files.

To use the standard .cshrc, use your favorite text editor to edit your own .cshrc, including the command

```
source /home/owl/public/master_cshrc
```

in the second or third line. All commands after this source add your own individual preferences such as aliases, private executable paths, et cetera. You should also clean up your .cshrc, removing any duplicitous information that is already covered in the standard version. Look at ~satogata/.cshrc, ~salty/.cshrc, and ~peggs/.cshrc for examples. BEWARE of making changes to the $path and $MANPATH variables that do not include the previous values; for example, if you want to add your own ~/bin directory to your default $path, include the line

```
set path=($path ~/bin)
```

in your .cshrc *after* sourcing the master.

To use the standard .login, use your favorite text editor again to edit your own .login, and include the command

```
source /home/owl/public/master_login
```

at the end of the file. The master_login file is rather verbose about what is available and not available on the particular system you are logging into - if you want to suppress most of these messages, include the line "setenv QUIET" before sourcing the master_login. For examples, see ~satogata/.login, ~salty/.login, and ~peggs/.login.

The master_login can also automatically start either OpenWindows or tvtwm for you, if you set the environment variable $mychoice before the "source" statement:

set mychoice = nowin	Start no windows (default)
set mychoice = twm	Start tvtwm
set mychoice = openwin	Start OpenWindows

Choosing twm uses the ~/.xinitrc file for X windows resources, while choosing openwin uses the ~/.owrc file for these resources. Sample copies of these files are also available in the /home/owl/public area.

Windowing

There are currently two window managers available on the zoo domain - OpenWindows (a Sun proprietary package using olwm, the OpenLook window manager) and tvtwm (a free X window manager). A few packages, such as AutoCAD, will only work under OpenWindows. Most others, including many of the tools available in OpenWindows, work in either window environment. Chris has written a couple of tiny scripts which will switch from one window manager to the other; however, they only work if you started up your windowing system with twm:

toow	Goes from twm window manager to OpenWindows
totwm	Goes from OpenWindows to twm window manager

OpenWindows is being phased out over the next few years by Sun, to be replaced by a more widely-distributed X package in development. The major impact of this is currently on program developers, who should endeavor to use X and InterViews libraries instead of the XView libraries supplied by Sun. The control system for RHIC will certainly NOT have the OpenWindows look and feel. RAPIDE graphical interfaces are currently based on X11R5 (the latest version of X), InterViews, an object-oriented C++ class library for graphics, and Glistks, the C++ graphics library used by the ISTK package. There is some interest in software development under other types of window systems. The strongest of these in terms of commercial support is Motif, which is not available on the zoo domain as of this writing.

Text editors

Three text editors - emacs, vi and textedit (under OpenWindows) - are available on the zoo domain in their latest versions. Of these, emacs and vi are commonly used as the editors of choice, and emacs has many niceties such as a tutorial, online help, and links into other programs. The emacs tutorial may be accessed by starting emacs and typing 'Control-h t'.

Documentation

Unix man pages are available on a great variety of items. The csh commands

```
man -k <keyword>     or       apropos <keyword>
```

list short one-line descriptions of commands found in the man pages that contain a given keyword. The $MANPATH environment variable lists the directories searched during a man listing. A hardcopy of a particular man page file may be obtained using ptroff:

```
ptroff -man -P<printer> <manpage>
```

This presumes that you know where the particular man page resides, knowledge which is not always apparent and may require some listing of directories listed in the $MANPATH variable. For example, to print the man pages for gcc (all 53 of them!):

```
ptroff -man /usr2/local/man/man1/gcc.1
```

A graphical tool, Xman, also exists.

Info (executable 'info') is a program for reading online documentation. This is different from reading man pages, because info documentation is organized into sections with extensive cross-referencing, much like a book. Emacs also has an info-reading mode which may be accessed with the emacs command 'M-x info'. Currently documentation in the info browser includes

info	The info browser
emacs	The extensible self-documenting text editor
gcc	The gnu C and C++ compilers
cpp	The C preprocessor
gdb	The gnu executable debugger

Online documentation is also present in postscript forms for many programs - typically these are manuals distributed with public domain software, or documents produced locally. These are all available in the directory

/home/owl/public/docs	Public documentation

A postscript document may be previewed on any machine running X using ghostscript (gs), ghostview or pageview. These documents are printable using lpr; many have already been printed and are available from Todd.

Backups and archiving

A pair of programs, fmtflop and mntflop, are in the /usr/local/bin directory; they provide user access to a Sparc IPX 3.5" disk drive (typically with about a megabyte of user storage per disk) without needing root mounting privileges. Here is a summary of commands:

fmtflop	Formats floppy currently in drive
mntflop <directory>	Mounts formatted floppy on directory
mntflop -u	Unmounts currently mounted floppy
eject	Ejects floppy from drive

Beware - the Sun floppy drives are notorious for sensor problems that claim the disk is write protected even when it is not. If you get an error like this, eject the disk, reinsert it and try again.

Disk space is a problem on occasion on our domain, especially for users on certain disks that seem to fill up on a regular basis. It is important for users to archive portions of their home directory that will lie dormant for a length of time, either to compressed archive files or to tapes. The unix tar and compress utilities can be used to accomplish this; however a more effective compression is obtained using gnutar, the gnu version of tar. To compress and tar a directory, use the command

```
gnutar -zcvf <directory.tar.gz> <directory>
```

Avoid using absolute pathnames for the directory specification, as gnutar preserves the directory hierarchy as it archives. This creates the file <directory.tar.gz>, and the extension labels it as a gnu compressed (gzipped) tar archive. The directory can then be deleted normally. To restore the director from the archive, type

```
gnutar -zxvf <directory.tar.gz> <directory>
```

To archive large sections of your directory onto magnetic tapes, or to retrieve files from the automatic ABARS optical disk backup that occurs daily, please contact either Judy Colman or Todd Satogata. Both exabyte (8 mm) and 1/4" tape drives are available on owl.

2 Controls environment

Shared project directories

The project directory structure that we use when developing shared code, such as ISTK or LAMBDA, is organized as follows. (This structure also works well for private work).

```
                              Project
                                 |
  |-----------------|------------|-------------------------|---------------|
bin              include        lib    *            MyApp          MyLib
                    |                  *              |
                 |--------|            *      |--------|-------|--------|
             MyLib   MyApp             *     RCS   SUN4  SGI    include
                                       *             |
                                       *           MyApp
                                       *             |
                                       *            RCS
                                       *
             Distribution Side         *     Development Side
```

Code development proceeds on the right hand side of this diagram. Source code lives in MyApp (and/or MyLib if appropriate), and headers live in MyApp/include/MyApp. Note that in all the public project systems, object and executable code is created in a directory *below* the source, typically called SUN4. This is prettier - your source is not clogged with masses of .o files. More importantly, it allows us to keep clean control of the code for different computers - at the moment just Sun-risc and VxWorks (68xxx real-time machines) but SGI will appear soon. The extra effort now will pay off real soon.

Compilation is under the control of the 'make' utility, which is smart enough to recognize the machine architecture, and places the executable in the appropriate SUN4, SGI, et cetera, directory. When the developer has reached a stable state of MyApp, he or she makes a new release available to the public by moving executables, header files, and library files to the left side of the diagram, for distribution. This migration is fully automated using the 'Makefile' utility (see below).

A second vital utility that is also described in more detail below is RCS, the Revision Control System. RCS tracks the changes made to a set of source code and header files, and allows more than one person to work on the source code simultaneously, without (unnecessary) confusion.

Makefiles

The Makefiles that you see in, for instance, the LAMBDA directories are quite complex. This is for good reason. Please use them as templates for code that is likely to be shared. When developing, either sit in the source directory Project/MyApp, and make using the command

```
make MyApp
```

or sit in the architecture directory, for example Project/MyApp/SUN4 and make by typing

```
make -f ../Makefile MyApp
```

This separates the object, executables and libraries without having source duplicated everywhere. If you prefer the latter style, it is useful to know that the command 'make -f ../Makefile' has been aliased as 'um' in the master .cshrc file. When you are ready to install distribute a new version of MyApp for public distribution, use the Makefile target 'install-all' by sitting in Project/MyApp and typing

```
make install-all
```

This will compile MyApp (and MyLib, and any other dependant libraries) and then move the resulting executables into Project/bin, libraries into Project/lib and includes into Project/include/MyApp.

Why, for Pete's sake? Well, the idea is that when the developer decides he has a testable version, he can put it in these directories and continue futzing with his development copies without blowing away anyone who is using his semi-installed headers et cetera. As his development stuff is all nicely RCS'd he can see what changes he makes to the semi-installed stuff; when all testers are satisfied, he may then automatically copy this stuff to the full installation place (usually /usr/local/bin, /usr/local/lib, /usr/local/include) with an understanding of what other dependant stuff needs recompiling. No? Well, it makes sense to us.

In addition to solving portability and distribution problems, the standard Makefiles address other difficulties by having a number of standard targets in them:

install-all:	will compile applications, libraries and header file and put them in the relevant place for *public* consumption. The default build targets should not install things, but just leave them locally so that you can work on code without potentially messing up other users.

tar:	This target builds a tar file of the source so it can easily be shipped elsewhere.
rcsinfo:	will produce information about who has what checked out.
build:	which creates architecture directories and libraries
clean-all:	gets rid of all executables, object and libraries so that a full make will happen.

Most of these do things recursively to subdirectories, which are explicitly defined in the Makefiles. For example, to install the whole of ISTK, you do

```
make install-all
```

in the top directory and everything gets built. Similarly,

```
make rcsinfo
```

at the top gives you information about who has what source checked out.

The Makefiles are set up so that they are driven from environment variables, so that changes in compiler or type of computer can be accommodated by using different setup files rather than editing the makefile or (argghhhhh!!) having several copies of it. Examples of the setup files are in $ISTKPLACE/ENV ($ISTKPLACE is one of the environment variables the Makefiles use); these setups go into effect by sourcing them from the c-shell. The makefile should also be constructed so that it will execute all these targets recursively through any source subdirectories.

It is convenient to check what architecture type you are on every time you create a shell. For instance, when Chris logs on to amnesia (a Californian HP), he automatically sets the architecture type environment variable ARCH to HP68K, and runs a shell script called gcc which sets up all the other required variables: EVERYTHING is defined, CC, CPLUS, AR, RANLIB, CP . Each shell script works for all architectures with no problem, differentiating by testing $ARCH. It is even possible to write the architecture type on the title bar of the shell window, to see what's what. If he logs on to ctrldev1, a Sun machine, Chris must decide whether to compile for the Sun or to cross-compile for VxWorks. By default he sets up for sun compilation, but can source the vwx script to work with VxWorks.

If you want to know more, type

```
man make ....
```

but it's probably easier to ask Todd or Chris.

RCS (Revision Control System)

This is to control the update of software which is likely to be shared; if you're just hacking together something for your own use, don't bother. RCS is sophisticated and full details are to be had by typing

```
man -k rcs
        ci (1)              - check in RCS revisions
        co (1)              - check out RCS revisions
        rcs (1)             - change RCS file attributes
        rcsdiff (1)         - compare RCS revisions
        rcsfile (5)         - format of RCS file
        rcsintro (1)        - introduction to RCS commands
        rcsmerge (1)        - merge RCS revisions
        rlog (1)            - print log messages and other information about RCS files
```

and by typing

```
man rcs
```

et cetera for details. However, for most work, the few aliases set up in the master_cshrc are sufficient to ease use. To start RCS control of (for example) some .c files, go to the directory with the source, make a directory called RCS, and type

```
rcsenter *.c
```

and answer the questions about description of the code. If you try to edit one of your files, you will find it set to read-only. DON'T CHANGE THE PERMISSIONS! Type

```
rcsedit myfile.c
```

and rcs will give you access. Edit away, and at relevant parts of your editing - when you've tested some of the changes, or when you're about to embark on a massive change - tell rcs with

```
rcsenter myfile.c
```

You'll be asked to comment on what you've just done; if you make an enormous mess of things, or if you need to know the history of changes, rcs will be able to reconstruct the source at each stage of the editing. To find out who has code checked out - presumably because they're working on it - type

```
rcsinfo
```

If someone *is* working on the code, don't copy it and hack it yourself as merging two versions can be tricky. Rather, find the person and persuade them to check the code in - and when you're finished, remember to check it in yourself. If you prefer to copy the code to your own area to work on it, CHECK IT OUT FIRST - then other people will be warned that you are doing something. To see the changes since the last check-in revision, type

```
rcsdiff myfile.c
```

Finally, to keep track of the rcs version inside your source code, add the line

```
/* $Header: $ */
```

at the top of each file; rcs will then fill in good information for each revision, along the lines of

```
/* $Header: /home/owl/salty/dial/RCS/waldo.cc,v 1.5 1993/08/17 16:35:33 salty Ex p salty $ */
```

LAMBDA directories

To illustrate this, look at the levels of directories in lambda: At the top (in $LAMBDA) we have mostly directories: (most of the files have been taken out of this listing to make it easier to understand)

```
Makefile        --- this controls all the lower level Makefiles
bin/            --- executables - but there are directories below
dbsf/           --- Now the directories that contain source for each application
demos/
docs/           --- Something to read
fiduce/
flatin/
gui/
include/        --- include files that other people may need - to use with libraries, for instance.
lib/
ramp/
spare/
survey/
sybase/
twiss/

bin:            --- The bin directory has more than executables:
Makefile
SUN4/
copy.in*        --- these executables are not machine specific - they are scripts
copy.out*
rnl*
strength.in*
strength.out*
```

```
bin/SUN4:           --- and here are binaries for sun-risc
dbsf*
dbwhatis*
dxf*

include:            --- public include files, put here after make install-all in the various application
                    directories.
DO_NOT_EDIT_THESE_FILES     --- this, too is a file. Don't.
fiduce.h
flatin.h
lattice_defines.h

lib:
SUN4/

lib/SUN4:
libflatin.a
liblambda.a

survey:
Makefile
RCS/
SUN4/
include/
survey.c            --- Finally, some source code - under RCS control.
survey.cmd
survey_func.c
survey_top.c

survey/RCS:
survey_top.c,v      --- RCS holds the base version and all the deltas so it can reconstruct the code at
                    any point in development.

survey/SUN4:        --- Here the object files and local executables and libraries are created.
survey*
survey.o
survey_func.o
survey_top.o

survey/include:     -- ...and local includes - NOT for public use.
RCS/
survey.h
survey_dict.h
survey/include/RCS:
survey.h,v
survey_dict.h,v
```

ISTK (Integrated Scientific Tool Kit)

This package contains stuff to describe data as it moves from program to program, and ways of controlling the data flow via events both inside and between programs. An introduction to the system is in $ISTKPLACE/Doc/Glossy.ps - use ghostview to peruse it. Alternatively, Chris will flame on at the drop of a hat. Some of the ISTK executables available now are:

kaspar	A fast graphical browser for SDS datasets...
asiv	...and its pre-analyser which finds what it can plot
conv	A utility for conversion of SDS binary data between different machine architectures
db2sds	For generating C-structures and their header files from Sybase SQL queries
f2sm	To move an SDS dataset from file to shared memory ...
sm2f	... or from shared memory to file
fft	For doing fft's on bits of SDS datasets
fstriphead	For separating SDS headers from their data
gasp	A pre-processor to ease generation of SDS datasets from C code
glish	The glish executive for process sequencing ...
glishmon	...and its monitor to show what's going on.
latview	For producing mimic diagrams of the machine lattices, and generating positional events to send to other programs.
lino	For SDS dataset selection.
sds2a	Generates ascii files from SDS datasets
sds2db	Sticks SDS described C structures into sybase
sds2ex	Generates simple ascii files from SDS datasets that can be read into excel, et cetera.
sid	A textual browser for SDS.
smd	Dumps SDS dataset definitions and data contents to stdout.
srm	Removes shared memory datasets
template	Generates control templates for SDS datasets (for example, to set editing limits or locks on particular data fields)
wshm	Tells you what shared memory datasets you have
cleanshm	Cleans out all shared memory datasets.

Holy_Lattice

It is RAP's responsibility to organize shared directories for information serving, as well as for software development. For example, the area /usr/local/Holy_Lattice has recently been described in an AP note RHIC/AP/6 . This is where we want those responsible for declaring 'this is the currently official lattice' to leave all the commonly used files; so that

1. Your normal user doesn't have to ask what is the official lattice
2. That person does not have to know how to invoke dbsf, survey etc etc

The file structure is as follows:

```
                          /usr/local/Holy_Lattice
                                     |
  |--------------------|----------------|------------------|---------------|----------------|
Blue              BTransfer       Yellow          YTransfer       Build*            mfile
```

Within each directory is created the files:

```
<lattice-name>          <---- flat sds
<lattice-name>.dxf      <---- dxf file for AutoCad
<lattice-name>List      <---- list sds for latview
Survey                  <---- survey sds (RHIC coordinate frame)
Twiss                   <---- twiss sds
end.top                 <
plan.top                <---- topdrawer files
side.top                <
survey.asc
survey.log
three_d
twiss.asc
twiss.log
```

where <lattice-name> is the same as the directory name; that is, one of Blue,Yellow,BTransfer or YTransfer. Such a file list is made by typing:

```
Build <database-name> <beamline>
```

in the top directory. For example

```
Build ags_to_rhic a2blue
Build rhic92r0p3 blue
```

If any more files should be automatically generated, ask Chris or Todd or do it yourself by changing the mfile makefile in the top directory.

Sybase database access

Getting access to the database from programs can cause difficulties. In particular, it is not a good idea to compile in login information such as user names, password of servers - these things have a habit of changing and anyway the Powers That Eb would be most upset to see passwords written en clair in accessible source code. A way out of this has been invented, and tested to a certain extent. It is in the function db_login() in libdbsds.a . The strategy is as follows: the call is used as:

```
char *host,*progname,*user,*password;
dbproc = db_login(host,progname,user,password);
```

'progname' is usually simply argv[0], passed into main() at the start of the program. If this string is too long for sybase (for example, when running under gdb the program name is the complete pathname) db_login() will use only the last component of the path.

If any of the remaining pointers are NULL, or point to a NULL string, values will be searched for. First, a file called $HOME/.dbinit will be opened. If it exists, it will be searched for lines like:

```
password hello
user gremlin
host SYBASE491
```

The first word is the keyword, separated by a space from the value. If that fails, the file $ISTKPLACE/.dbinit will be opened; you'll get logged on as lattice_reader. Failing this, 'user' will be set to the current login name, and 'host' will be taken from the DSQUERY environment variable.

Deliberate omissions

We don't talk about graphics. ISTK graphics, running on top of InterViews, is there if you want simple panels. It works reasonably well and there are some good applications. It will be superseded in X11 revision 6 by an X graphics class library, which is the way to go. In the meantime the really important parts of ISTK - data control and event flow within and without programs - is (almost) completely independent of a particular graphics system.

We also don't talk about FORTRAN or C++, although most of the tools mentioned apply to these languages as well as they do to C. The assumption is that most of the work done by RAP will be in C, although some will be done in C++, while FORTRAN will be tolerated but not (necessarily) fully supported.

3 General environment

Compilers and libraries

The Fortran77 compiler, f77, is available, as well as the sun C compiler (cc) and version 2.4.5 of the gnu C and C++ compiler (gcc). All execution and path defaults are installed by the master_cshrc. Unless otherwise specified, gcc was used to compile all libraries mentioned below. gcc is also the standard compiler used for RAP controls software development.

Various developers libraries are available on the zoo domain, including X libraries (in a variety of places - a sad fact which will soon be corrected), InterViews, ISTK and libg++ libraries. In the future, other libraries such as Numerical Recipes and Cern Lib will be included in this listing. The header files for these libraries are in several places, specified for searching with the compile directive -I:

/usr2/local/include/X11	X libraries
/usr2/local/include/InterViews	InterViews libraries
/usr2/local/ISTK/include	ISTK libraries
/usr2/local/lib/g++-include	libg++ libraries

The actual libraries themselves are in several places as well; these are specified at link time with the compiler directive -L:

/usr2/local/lib	X libraries, InterViews and libg++ libraries
/usr2/local/ISTK/lib/$ARCH	ISTK libraries

Plotting and postscript

There are four plotting packages in fairly common use on the zoo domain:

sm	SuperMongo
gnuplot	Gnu plotting routines
xvgr	XView graphics plotter
xfig	X distribution utility
td, topdraw	TopDrawer graphics package

SuperMongo (executable: sm) has extensive documentation, both in emacs info and online.

Gnuplot is a plotting program distributed by gnu, simple and straightforward and small; it includes online documentation as well as a man page.

Xvgr is based on the XView toolkit provided by Sun OpenWindows; it is highly interactive, and good for producing high quality plots with relatively little strain.

Xfig is part of the standard X distribution. It allows you to draw figures and lines easily and interactively on any X terminal, and is useful for quick sketching of diagrams and figures. It can output both normal and Encapsulated PostScript.

TopDrawer is a publication quality plotting package from Stanford, originally designed for use in high energy experimental physics. It deserves some specific mention because of its installation on this system. Because all the other packages produce PostScript output (for including plots in publications, for example), another variant of TopDrawer, called topdraw, was installed on the system to produce postscript plots (named topdraw.ps) from a TopDrawer input format ascii file. Four executables are present in /usr2/local/bin:

td	X topdrawer graphics; does not produce postscript
topdraw	TopDrawer graphics conversion to PostScript
tdp	TopDrawer graphics in portrait PostScript
tdl	TopDrawer graphics in landscape PostScript

Note that the PostScript produced by topdraw is not Encapsulated Postscript - that is, certain things must be added to the PostScript produced by topdraw in order for it to be compatible with various TeX and DVI utilities, such as psfig. A man page is available for the "topdraw", and documentation on TopDrawer is available from Steve, Todd, or the CCD Software and Documentation Store.

Three PostScript previewers are available: GhostScript (gs), Ghostview (which adds a graphical user interface to ghostscript) and Pageview,which is part of the Sun XView distribution. All three previewers work under both twm and olwm window managers, and both Ghostview and Pageview allow users to select pages to view interactively. These previewers are particularly useful for previewing large manuals and reference documents listed in the /home/owl/public/docs and/home/owl/public/rhic_ap_notes areas. This saves paper over the alternative, which is to simply print them out before deciding whether or not a hardcopy is needed.

Symbolic algebra

Mathematica, a popular symbolic algebra and graphics package, is not available on the zoo domain. It is available on the CCD unix machine ax61.bnl.gov, as described in the sifview under 'mathematica'. [See sifview in the section CCD Software.]

Maple, another symbolic algebra package, is available on all zoo domain machines. It can be executed from a text window, such as a vt100 or a vt220 terminal, by typing

```
maple
```

The X version of maple is run by typing

```
xmaple
```

Maple documentation and online help are available. Maple supports 2d and 3d graphics, including various lighting patterns and mesh graphics. Maple also has a large set of programming and function libraries for analysis and computation, as well as symbolic manipulation.

CCD software

Software distributed from CCD is listed in a program called sifview (Software Information File Viewer). This program provides an X-based viewer to get the latest information on software available from CCD, including information on mount points and environment variables to set. It is run by typing

```
sifview
```

The files this software views are also present in ASCII form in the directory

```
/ssg1/software1.bnl.gov/common/sif
```

If you are interested in any of the CCD software that is available for platforms on the zoo domain, and it is not already installed, contact Todd Satogata (x5452) or Judy Colman (x2590). Much of this software, though not all, already has appropriate mount points in place for use on Sun workstations.

Usenet news

Usenet news is available from nodes owl, bear and bunny. (Other machines can easily be added; use of the Usenet news server, bnlux1, requires machine registration.) There are two ways to read news:

xvnews	News reader based on XView/OpenWindows tools
emacs -f gnus	Gnus, the emacs news reader

Gnus may also be started from inside emacs by typing `M-x gnus'. There are approximately 2500 news groups covered by the server at bnlux1,including both recreational and discussion groups, and serious research groups. Recently, at the behest of Robert Ryne and others, the newsgroup sci.accelerators

was added to the usenet group list. This will likely be the future method for quick and timely updates on events in the accelerator and physics community. Other interesting newsgroups include

sci.physics	physics
sci.nonlinear	nonlinear dynamics

and a host of bnl specific groups (bnl.*).

Sybase

Sybase, with a database engine running on vermeer.cad.bnl.gov, is the relational database of choice at present for users working on the zoo domain. All paths are correctly set in the master_cshrc, and other relevant environment variables for the LAMBDA distribution are described in the LAMBDA manual located in /home/owl/public/docs. To use Sybase on your own, a login is required; Sharon Spark (x4111) handles this, as well as Sybase administration questions while Susan Wong is on maternity leave.

The current primary use of Sybase is to store the lattice and optics database for revisions of RHIC lattice design, including injection lines from AGS to RHIC. These databases are named

ags_to_rhic	AGS to RHIC injection lines
rhic92r0p2	RHIC 92 lattice, revision 0.2
rhic92r0p3	RHIC 92 lattice, revision 0.3
rhic_2	???

Access to these databases, and conversion to several standard lattice design programs such as MAD, TEAPOT, SYNCH, et al, is provided by the tools in LAMBDA. For more information please contact Steve or Todd, or peruse the documentation.

Metafont, TeX, LaTeX and dvi

The TeX typesetting package, used to typeset and format technical documents in a professional manner, is installed locally on owl. This distribution has several improvements over the CCD distributed version, and includes

tex	Knuth's basic TeX
mf	Knuth's Metafont font design and creation program
latex	Leslie Lamport's LaTeX
texsis	TeX Extensions from the university of Texas.
xdvi	X dvi file previewer

dvips	Produce PostScript files or printed copy from a DVI document, produced by TeX or LaTeX.

Man pages and documentation are available for all of the above, including books written by Knuth about TeX and Metafont. Sample files abound, as well as document styles designed for the formats of several major conference proceedings and publications. All environment variables are set to reasonable default values on the cluster - you need not worry about them in your .cshrc file unless you want to use your own style files or fonts.

Liken

Liken, a Macintosh native executable emulator running under X, is also available on all nodes of the zoo domain, though at the current time only two floating licenses are available. The current implementation of Liken supports Mac floppy disks and most desk tools, but does not support full System 7 compatibility or color displays. An upgrade which supports networking is expected to arrive by August 23, with System 7 compatibility arriving within 1-2 months afterwards. Liken environment variables are set up in the master_cshrc file. It may be started by typing

```
Liken -boot=/home/owl/mac/disks_1/boot_disk
```

Available software includes MS Word for word processing, and other packages.

Lattice design and tracking tools

See the area /usr/local/lattice_tools .