The Top Level Makefile

Timothy Daly

March 24, 2017

Abstract

Contents

1 The overall build process

Makefiles are responsible for building all of their subdirectories and then themselves. The system is built in the following order

1. books – building pdf files

2. src/scripts – building scripts

3. src/lib – building lib files

4. lsp – building gcl common lisp

5. src – building src

6. src/interp – building interpreter files

7. src/share – building share files

8. src/algebra – building help files

9. src/algebra – building input files

10. src/algebra – building xhtml files

11. src/algebra – building algebra files

   - algebra bootstrap complete
   - layer 0 of 24 complete
   - layer 1 of 24 complete
   - layer 2 of 24 complete
   - layer 3 of 24 complete
   - layer 4 of 24 complete
   - layer 5 of 24 complete
   - layer 6 of 24 complete
   - layer 7 of 24 complete
   - layer 8 of 24 complete
   - layer 9 of 24 complete
   - layer 10 of 24 complete
   - layer 11 of 24 complete
   - layer 12 of 24 complete
   - layer 13 of 24 complete
   - layer 14 of 24 complete
   - layer 15 of 24 complete
   - layer 16 of 24 complete

- layer 17 of 24 complete
- layer 18 of 24 complete
- layer 19 of 24 complete
- layer 20 of 24 complete
- layer 21 of 24 complete
- layer 22 of 24 complete
- layer 23 of 24 complete
- layer 0 copy complete

12. src/etc – building etc

13. src/doc – building doc files

14. src/input – running regression tests

2 General Makefile Structure

Makefiles are responsible for four things. First, they have to set up the output directory structure so that all of the build machinery can assume it exists. Second, they have to build all of the files in their own directory. Third, they have to invoke Make on each of their subdirectories. This forms a natural tree walk of the directory structure. Fourth, they have to explain all of the details about the directory, the files it manages and its subdirectories.

The clean stanza has been modified to be more effective. Previously it walked the Makefile hierarchy trying to clean subdirectories. This method often fails for various reasons (e.g. permissions, incomplete builds, etc). Now we simply remove the created files directly.

2.1 The Top Level Makefile

We have added a stanza to separate the build of documents from the build of source files. As much as possible we would like to do the document builds in parallel with the source builds. We kick off a make for the documents in the background and then kick off a make in the foreground for the source code. This is independent of using make in parallel and instead uses the shell to fork the processes. We have not had much luck getting make to build in parallel reliably.

Note that make cannot handle recursively calling itself in the same directory so we have to expand the serial forms inline. Cheesy.

— parallel —

```
all: rootdirs axiom.sty tanglec libspad
@ echo 1 making a ${SYS} system, PART=${PART} SUBPART=${SUBPART}
@ echo 2 Environment ${ENV}
@ ${BOOKS}/tanglec Makefile.pamphlet "Makefile.${SYS}" >Makefile.${SYS}
@ cp ${BOOKS}/dvipdfm.def ${MNT}/${SYS}/doc
@ cp ${BOOKS}/changepage.sty ${MNT}/${SYS}/doc
@ ${EXTRACT} Makefile.pamphlet
@ cp Makefile.pdf ${MNT}/${SYS}/doc/src/root.Makefile.pdf
@ if [ "${RUNTYPE}" = "parallel" ] ; then \
   ( echo p4 starting parallel make of input files ; \
     ${ENV} ${MAKE} input ${NOISE} & ) ; \
  else \
          if [ "${BUILD}" = "full" ] ; then \
   ( echo s4 starting serial make of input files ; \
           cd ${MNT}/${SYS}/doc/src/input ; \
     cp ${BOOKS}/axiom.sty . ; \
           cp ${SRC}/input/*.eps . ; \
     for i in `ls ${SRC}/input/*.input.pamphlet` ; do \
      if [ .${NOISE} = . ] ; \
      then \
            latex $$i ; \
      else \
       ( echo p4a making $$i ; \
         latex $$i >${TMP}/trace ) ; \
      fi ; \
           done ; \
     rm -f *~ ; \
     rm -f *.pamphlet~ ; \
     rm -f *.log ; \
     rm -f *.tex ; \
     rm -f *.toc ; \
     rm -f *.aux ) ; fi ; \
  fi
@ if [ "${RUNTYPE}" = "parallel" ] ; then \
    ( echo s2 starting parallel make of books ; \
      echo s3 ${SPD}/books/Makefile from \
                 ${SPD}/books/Makefile.pamphlet ; \
```

```
        cd ${SPD}/books ; \
              ${EXTRACT} Makefile ; \
              cp Makefile.pdf ${MNT}/${SYS}/doc/src/books.Makefile.pdf ; \
        ${ENV} ${MAKE} & ) ; \
  else \
    ( echo s2 starting serial make of books ; \
      echo s3 ${SPD}/books/Makefile from \
                  ${SPD}/books/Makefile.pamphlet ; \
      cd ${SPD}/books ; \
              ${EXTRACT} Makefile ; \
              cp Makefile.pdf ${MNT}/${SYS}/doc/src/books.Makefile.pdf ; \
              if [ "${BUILD}" = "full" ] ; then \
        ${ENV} ${MAKE} ; fi ) ; \
  fi
@ echo p7 starting make of src
@ ${ENV} $(MAKE) -f Makefile.${SYS}
@ echo 3 finished system build on `date` | tee >lastBuildDate

libspad:
@ echo 11a making libspad
@ ( cd ${OBJ}/${SYS}/lib ; \
    ${BOOKS}/tanglec ${BOOKS}/bookvol8.pamphlet Makefile >Makefile ; \
            ${ENV} ${MAKE} libspad.a )

axiom.sty:
@ echo 11c copying books/axiom.sty
@ cp ${BOOKS}/axiom.sty .

rootdirs:
@ echo 11 checking directory structure
 mkdir -p ${TMP}
 mkdir -p ${INT}/algebra
 mkdir -p ${INT}/hyper
 mkdir -p ${INT}/input
 mkdir -p ${INT}/interp
 mkdir -p ${INT}/sman
 mkdir -p ${OBJ}/${SYS}/bin
 mkdir -p ${OBJ}/${SYS}/etc
 mkdir -p ${OBJ}/${SYS}/graph
 mkdir -p ${OBJ}/${SYS}/hyper/pages
 mkdir -p ${OBJ}/${SYS}/interp
 mkdir -p ${OBJ}/${SYS}/lib
 mkdir -p ${OBJ}/${SYS}/sman
 mkdir -p ${OBJ}/${SYS}/proofs
 mkdir -p ${MNT}/doc
 mkdir -p ${MNT}/${SYS}/algebra
 mkdir -p ${MNT}/${SYS}/src/algebra
 mkdir -p ${MNT}/${SYS}/autoload
 mkdir -p ${MNT}/${SYS}/bin
 mkdir -p ${MNT}/${SYS}/doc/axbook
```

```
    mkdir -p ${MNT}/${SYS}/doc/bitmaps
    mkdir -p ${MNT}/${SYS}/doc/hypertex/bitmaps
    mkdir -p ${MNT}/${SYS}/doc/msgs
    mkdir -p ${MNT}/${SYS}/doc/ps
    mkdir -p ${MNT}/${SYS}/doc/cookbook
    mkdir -p ${MNT}/${SYS}/doc/spadhelp
    mkdir -p ${MNT}/${SYS}/doc/src/input
    mkdir -p ${MNT}/${SYS}/graph/parabola.view
    mkdir -p ${MNT}/${SYS}/input
    mkdir -p ${MNT}/${SYS}/lib/graph
    mkdir -p ${MNT}/${SYS}/lib/scripts

input:
@ echo p9 making input documents
@ if [ "${BUILD}" = "full" ] ; then \
  ( cd ${MNT}/${SYS}/doc/src/input ; \
    cp ${BOOKS}/axiom.sty . ; \
    for i in `ls ${SRC}/input/*.input.pamphlet` ; \
            do latex $$i ; \
            done ; \
    rm -f *~ ; \
    rm -f *.pamphlet~ ; \
    rm -f *.log ; \
    rm -f *.tex ; \
    rm -f *.toc ; \
    rm -f *.aux ) ; fi
```

——————-

—— * ——

```
\getchunk{ENVDEFAULTS}
\getchunk{ENVAR}
\getchunk{parallel}
\getchunk{book}
\getchunk{tanglec.c}
\getchunk{literate commands}
\getchunk{install}

document:
@ echo 4 making a ${SYS} system, PART=${PART} SUBPART=${SUBPART}
@ echo 5 Environment ${ENV}
@ ${BOOKS}/tanglec Makefile.pamphlet "Makefile.${SYS}" >Makefile.${SYS}
@ ${ENV} $(MAKE) -f Makefile.${SYS} document
@echo 6 finished system build on `date` | tee >lastBuildDate

clean:
@ echo 7 making a ${SYS} system, PART=${PART} SUBPART=${SUBPART}
@ echo 8 Environment ${ENV}
```

```
@ rm -f axiom.sty
@ rm -f Makefile.pdf
@ rm -f Makefile.dvi
@ rm -f Makefile.${SYS}
@ rm -f books/Makefile
@ rm -f books/Makefile.dvi
@ rm -f books/Makefile.pdf
@ rm -f books/axiom.bib
@ rm -f lsp/axiom.sty
@ rm -f lsp/Makefile lsp/Makefile.dvi lsp/Makefile.pdf
@ rm -rf lsp/gcl*
@ rm -f src/axiom.sty
@ rm -f src/Makefile
@ rm -f src/Makefile.dvi
@ rm -f src/Makefile.pdf
@ rm -f src/algebra/book*pamphlet
@ rm -f src/algebra/*.spad
@ rm -f src/algebra/Makefile
@ rm -f src/etc/axiom.sty
@ rm -f src/etc/Makefile src/etc/Makefile.dvi src/etc/Makefile.pdf
@ rm -f src/interp/book*pamphlet
@ rm -f src/interp/axiom.sty
@ rm -f src/input/axiom.sty
@ rm -f src/share/axiom.sty
@ rm -f src/share/Makefile src/share/Makefile.dvi
@ rm -f src/share/Makefile.pdf
@ rm -rf int
@ rm -rf obj
@ rm -rf mnt
@ for i in `find . -name "*~"` ; do rm -f $$i ; done
@ rm -f lastBuildDate
@ rm -f books/tanglec
@ rm -f src/input/Makefile src/input/Makefile.dvi
@ rm -f src/input/Makefile.pdf
@ rm -f src/interp/Makefile src/interp/Makefile.dvi
@ rm -f src/interp/Makefile.pdf
@ rm -f src/share/Makefile src/share/Makefile.dvi
@ rm -f src/share/Makefile.pdf
@ rm -f Makefile.aux
@ rm -f Makefile.log
```

————-

2.2 Environment

2.2.1 VERSION

The VERSION variable is a unique string intended to show up in the banner at
startup time. I can be anything but is intended to be a unique way of identifying

the CVS version so we can reference bug reports to versions.

The VERSION variable is used in the src/interp/Makefile to set a lisp variable boot::*build-version*. This variable is used by the `yearweek` function to construct the banner.

The banner also contains a build timestamp so we can determine when the image is compiled. We touch the file called `${MNT}/${SYS}/timestamp` and using a formatted form of its file information. See the YEARWEEK variable in the src/interp/Makefile.pamphlet and the `yearweek` function in src/interp/util.lisp.pamphlet.

2.2.2 SPAD

The SPAD environment variable is normally specified. It is expected to be a path to the top level directory of the shipped system. For example, if we want to build a linux system the SPAD variable should look like:

```
'pwd'/new/mnt/{\bf linux}
```

2.2.3 SYS

From the SPAD variable we look at the last directory name and create a version of Axiom for that system. The SYS environment variable is the last directory name in the SPAD variable.

2.2.4 SPD

The SPD variable is taken to be the current working directory where this Makefile lives. This is obviously the root of the whole system source tree. All Makefiles form environment variables based on this value.

Next we see the six top-level directories discussed above being defined using the SPD variable.

2.2.5 LSP

This variable specifies where the LSP subdirectory lives. It is normally a directory at the top level of the system but we do not assume that to be true. Other lisps might require it to be elsewhere.

2.2.6 GCLDIR

This file contained the only mention of the AKCLDIR variable which gives the path to the version of AKCL. Now that the system is running on GCL this variable has been renamed to GCLDIR. This cannot be eliminated entirely because the system uses this variable to look up a file called collectfn.lsp which is part of the GCL distribution. This file lookup is in conditional lisp code so other lisps will not see the file load. The collectfn.lsp code is used by GCL to generate the ".fn" files which are used to optimize function calling.

Also, lsp/sys-proclaims.lisp is a file generated during the GCL build which contains type information about lisp functions, allowing fast-function calling behavior.

When defining the environment, the SPD variable is defined as the current directory. SYS is taken as the last non-directory part of the environment variable `$AXIOM` (e.g. if `$AXIOM=/(a-path)/mnt/linux` then SYS=linux). It is **mandatory** that `$AXIOM` does **not** contain any trailing slash, because the `notdir` function will return the string following the final slash and would thus return the empty string.

2.2.7 SRC

The SRC subdirectory is a hand-generated, read-only top level directory containing the source code. This is assumed to be completely system-independent and, in general, it can reside on a CD or NFS mounted file system. This is useful for building several different kinds of systems (as specified by the SYS variable from a single source tree.

2.2.8 INT

The INT subdirectory is a machine-generated, system-independent top-level directory containing source code. Axiom builds from literate sources. This work only needs to be done once at the first build. The INT directory is a cache of work. It can be erased at will. However steps such as generating lisp code from spad code, while done by machine, are system-independent. Therefore this subdirectory, once built, can reside with the SRC subdirectory on CD or NFS as a read-only branch.

2.2.9 OBJ

The OBJ subdirectory is a machine-generated, system-dependent top level directory containing things like compiler binaries. The OBJ directory is a cache of work. It can be erased at will. Because it is system-dependent it needs to be written at build time by compilers for each specific system.

2.2.10 MNT

The MNT subdirectory is a complete, working copy of Axiom. This directory contains everything that is needed to run Axiom and can be copied anywhere. Everything in this directory takes its required information from the `$AXIOM` shell variable. Once this directory is copied the SRC, INT, and OBJ subdirectories can be erased.

2.2.11 ZIPS

The ZIPS subdirectory contains particular versions of subsystems that Axiom needs in tar-gzip format. The Makefiles will unpack them. It also contains

patch files to these subsystems. The Makefile will apply those patches. Then it will configure and build the required subsystems.

2.2.12   TMP

The TMP directory is used in place of normal unix tmp in order to avoid writing outside of our build tree.

Note that TMP is a workspace in the OBJ directory. It is working space for temporary files since we cannot assume that we can write outside our own tree. Output from commands like the `document` command will generally be written to the TMP/trace file. If the build seems to hang while making a document file then check this file. It will contain the output of the latex command and the likely error in the tex file.

2.2.13   SPADBIN

The SPADBIN directory is the path to the executable binary directory of the shipped system. The directory contains all of the executable commands, such as the `document` command. The `document` command lives in the SRC/SCRIPTS subdirectory and will be copied to SPADBIN before we start walking the build subtree.

2.2.14   INC

The INC directory contains all the include files for the C programs.

2.2.15   The NOISE variable

The NOISE variable is used in the calls to the document command. In general, where the document command is called in the Makefiles it is called with the following form:

```
${SPADBIN}/document ${NOISE} foo
```

with the default value of NOISE being:

```
NOISE="-o ${TMP}/trace"
```

The reason NOISE exists is that the latex command will generate a page of output which is uninteresting during the make. However if there is a latex syntax error in a pamphlet file the make will continue past the error due to the nonstopmode flag. To see the actual error message rerun the make as:

```
make NOISE=
```

2.2.16 PART and SUBPART

Because of the size of this build we do everything possible to minimize the work necessary to rebuild. In order to allow finer control of the build we have two options that can be specified. The first is the PART variable. The second is the SUBPART variable. The PART variable basically specifies which directory we wish to build.

Setting the PART as:

```
PART=foo
```

will look for a stanza in the Makefile as:

```
\${PART}dir
```

which expands to:

```
foodir
```

Variable PART can be specified (environment or command-line) as one of:

```
(all | lib | install | lisp | interp | comp | graph | hyper
     | input | sman | boot | include | doc | algebra )
```

It is possible to be more specific with a directory using SUBPART.

2.2.17 DESTDIR and COMMAND

The install directory is /usr/local/axiom by default but this can be changed on the command line by typing:

```
make DESTDIR=/yourabsolutepath COMMAND=fullPathAndCommand install
```

The COMMAND string has been modified to use the DESTDIR variable so we can properly find the axiom command.

The DOCUMENT variable is now set to replace the direct call to the document command. This will allow it to be changed on the command line.

3 The Environment

— ENVDEFAULTS —

```
VERSION:="Axiom (May 2016)"

##### special paths
SPD:=$(shell pwd)
SRC:=${SPD}/src
LSP:=${SPD}/lsp
INT:=${SPD}/int
OBJ:=${SPD}/obj
```

```
MNT:=${SPD}/mnt
TMP:=${OBJ}/tmp
ZIPS:=${SPD}/zips
BOOKS:=${SPD}/books
SRCDIRS:="interpdir sharedir algebradir etcdir docdir \
          graphdir smandir hyperdir browserdir inputdir"

SYS:=$(notdir $(AXIOM))
DAASE:=${SRC}/share
PROOFS:=${OBJ}/${SYS}/proofs
SPAD:=${SPD}/mnt/${SYS}

SPADBIN:=${MNT}/${SYS}/bin
DOCUMENT:=${SPADBIN}/document
EXTRACT:=${BOOKS}/extract

##### installation paths
DESTDIR:=/usr/local/axiom
COMMAND:=${DESTDIR}/mnt/${SYS}/bin/axiom

##### functions we need
AWK:=gawk
PATCH:=patch
RANLIB:=ranlib
TAR:=tar
TOUCH:=touch
UNCOMPRESS:=gunzip

##### lisp related variables
BYE:=bye
\getchunk{GCLVERSION}
GCLDIR:=${LSP}/${GCLVERSION}
\getchunk{GCLOPTS-LOCBFD}
LISP:=lsp

##### C related variables
INC:=${SPD}/src/include
PLF:=LINUXplatform
CCF:="-O2 -fno-strength-reduce -Wall -D_GNU_SOURCE -D${PLF} -I/usr/X11/include -std=gnu89"
CC:=gcc
XLIB:=/usr/X11R6/lib
#LDF:=" -L/usr/X11R6/lib -L/usr/lib ${XLIB}/libXpm.a -lXpm"
LDF:=" -L/usr/X11R6/lib -L/usr/lib  -lXpm"
O:=o

##### command line control
NOISE:="-o ${TMP}/trace"
PART:= cprogs
SUBPART:= everything
RUNTYPE:=serial
```

```
# can be richtests, catstests, regresstests (see src/input/Makefile)
# alltests, notests
TESTSET:=notests
BUILD:=full
ACL2:=
COQ:=
```

—————-

3.1 The ENV variable

— ENVAR —

```
ENV:= \
ACL2=${ACL2} \
AWK=${AWK} \
BOOKS=${BOOKS} \
BUILD=${BUILD} \
BYE=${BYE} \
CC=${CC} \
CCF=${CCF} \
COMMAND=${COMMAND} \
COQ=${COQ} \
DAASE=${DAASE} \
DESTDIR=${DESTDIR} \
DOCUMENT=${DOCUMENT} \
EXTRACT=${EXTRACT} \
GCLDIR=${GCLDIR} \
GCLOPTS=${GCLOPTS} \
GCLVERSION=${GCLVERSION} \
INC=${INC} \
INT=${INT} \
LDF=${LDF} \
LISP=${LISP} \
LSP=${LSP} \
MNT=${MNT} \
NOISE=${NOISE} \
O=${O} \
OBJ=${OBJ} \
PART=${PART} \
PATCH=${PATCH} \
PLF=${PLF} \
PROOFS=${PROOFS} \
RANLIB=${RANLIB} \
RUNTYPE=${RUNTYPE} \
SPAD=${SPAD} \
SPADBIN=${SPADBIN} \
```

```
SPD=${SPD} \
SRC=${SRC} \
SRCDIRS=${SRCDIRS} \
SUBPART=${SUBPART} \
SYS=${SYS} \
TANGLE=${TANGLE} \
TAR=${TAR} \
TESTSET=${TESTSET} \
TMP=${TMP} \
TOUCH=${TOUCH} \
UNCOMPRESS=${UNCOMPRESS} \
VERSION=${VERSION} \
WEAVE=${WEAVE} \
XLIB=${XLIB} \
ZIPS=${ZIPS}
```

———-

3.2   book

This stanza constructs the book from the original pamphlet file. At this time
there is no difference between the pamphlet file and straight latex (intentionally).
Thus we just need to make sure the correct directories are in place, copy the
files, and run latex over the pamphlet file.

— **book** —

```
book:
@ echo 79 building the book as ${MNT}/${SYS}/doc/book.dvi
@ cp ${SRC}/doc/book.pamphlet ${MNT}/${SYS}/doc
@ cp -pr ${SRC}/doc/ps ${MNT}/${SYS}/doc
@ (cd ${MNT}/${SYS}/doc ; \
         if [ .${NOISE} = . ] ; then \
   ( latex book.pamphlet --interaction nonstopmode ; \
     latex book.pamphlet --interaction nonstopmode ) ; \
  else \
   ( latex book.pamphlet --interaction nonstopmode >${TMP}/trace ; \
     latex book.pamphlet --interaction nonstopmode >${TMP}/trace ) ; \
 fi ; \
 rm book.pamphlet ; \
 rm book.toc ; \
 rm book.log ; \
 rm book.aux )
@ echo 80 The book is at ${MNT}/${SYS}/doc/book.dvi
```

———-

16

3.3 tanglec.c

<div align="center">— tanglec.c —</div>

```
tanglec: books/tanglec.c
@echo t01 making tanglec from books/tanglec.c
@( cd books ; gcc -o tanglec tanglec.c )
```

<div align="center">———-</div>

3.4 src

We should recompile the world with the .fn information but not here.

```
( for i in `find . -name "*.lsp"` ; \
    do echo $$i ; touch $$i ; done )
( for i in `find . -name "*.lisp"` ; \
    do echo $$i ; touch $$i ; done )
@echo 15a remaking ${SRC}/interp for performance
@(cd src ; ${ENV} ${MAKE} )
```

<div align="center">— src —</div>

```
srcdir: ${SPD}/src/Makefile
@echo 15 making ${SPD}/src
@( cd src ; ${ENV} ${MAKE} )

${SPD}/src/Makefile: ${SPD}/src/Makefile.pamphlet
@echo 16 making ${SPD}/src/Makefile from ${SPD}/src/Makefile.pamphlet
( cd src ; \
   ${EXTRACT} Makefile ; \
   cp Makefile.pdf ${MNT}/${SYS}/doc/src/src.Makefile.pdf )

libspadclean:
@echo 17 cleaning ${OBJ}/${SYS}/lib
@rm -rf ${OBJ}/${SYS}/lib
@( cd src ; ${EXTRACT} Makefile )
@( cd src ; ${ENV} ${MAKE} clean )
@rm -f ${SPD}/src/Makefile ${SPD}/src/Makefile.dvi
```

<div align="center">———-</div>

3.5 src setup

<div align="center">— srcsetup —</div>

<div align="center">17</div>

```
srcsetup: ${SPD}/src/Makefile
@echo 18 making ${SPD}/src
@( cd src ; ${ENV} ${MAKE} setup )
```

———-

3.6   lsp

We delegate the details of constructing common lisp to the Makefiles in the
subtree. We need only ensure that the Makefiles are up to date.

When we first make GCL we the src/Makefile will create a marker file called
gcldir. This file has the same name as the stanza to create GCL and thus will
prevent GCL from rebuilding. We need to do this since we have no control over
the GCL makefiles.

The `${OBJ}/${SYS}/bin` directory is where the lisps get built.

3.6.1   LSPMakefile

We need to specialize the Makefile stanza based on the version of Lisp we plan to
use. At the moment there are 3 GCL versions which run Axiom: 2.4.1, 2.5, and
2.6 These are incompatible versions so we have different patches against them.
The details are in the lsp/Makefile.pamphlet file. Here we just decide which
section to choose. The default is 2.4.1 but if the GCLVERSION is changed then
the Makefile is rewritten to extract the later version. It looks for a chunk name
that matches the version number.

— **LSPMakefile** —

```
${LSP}/Makefile: ${LSP}/Makefile.pamphlet
@echo 20 making ${LSP}/Makefile from ${LSP}/Makefile.pamphlet
@( cd lsp ; \
 ${EXTRACT} Makefile.pamphlet ; \
 if [ "${GCLVERSION}" != "gcl-2.4.1" ] ; then \
 ${BOOKS}/tanglec Makefile.pamphlet ${GCLVERSION} >Makefile ; \
        fi ; \
 cp Makefile.pdf ${MNT}/${SYS}/doc/src/lsp.Makefile.pdf )
```

———-

Here we add mkdir -p `${OBJ}/${SYS}/lsp` because we need to rename the `gcl_collectfn.lsp`
back to collectfn.lsp. We also start adding support for sys-proclaim.lsp and other
dynamically collected proclaim information.

The obj/sys/bin dir is necessary to keep the compiled lisp image. The
obj/sys/lsp dir is necessary to keep collectfn and sys-proclaims. The collectfn.lsp
file is a special extension to GCL to collect type information during a compile-
file. This information gets written out to a .fn file. These .fn files can be loaded
and written out as a file containing proclaims information. If this proclaims in-
formation is available at compile time then the resulting function calls are much

more efficient. The sys-proclaims file contains type information about standard common lisp function calls.

<div align="center">— lsp —</div>

```
lspdir: ${LSP}/Makefile
@echo 19 making ${LSP}
@mkdir -p ${OBJ}/${SYS}/bin
@mkdir -p ${OBJ}/${SYS}/lsp
@echo ===================================
@echo lsp BUILDING GCL COMMON LISP
@echo ===================================
(cd lsp ; ${ENV} DESTDIR= ${MAKE} gcldir )
@(cp ${GCLDIR}/unixport/saved_gcl ${SPADBIN}/${GCLVERSION})

\getchunk{LSPMakefile}
lspclean:
@echo 21 cleaning ${OBJ}/${SYS}/ccl
@rm -rf ${LSP}/${GCLVERSION}
@rm -rf ${INT}/ccl
@rm -rf ${OBJ}/${SYS}/ccl
@rm -rf ${LSP}/gcldir
@rm -f ${LSP}/Makefile ${LSP}/Makefile.dvi
```

<div align="center">———-</div>

3.7   install

<div align="center">— install —</div>

```
install:
@echo 78 installing Axiom in ${DESTDIR}
@mkdir -p ${DESTDIR}
@cp -pr ${MNT} ${DESTDIR}
@echo '#!/bin/sh -' >${COMMAND}
@echo AXIOM=${DESTDIR}/mnt/${SYS} >>${COMMAND}
@echo export AXIOM >>${COMMAND}
@echo PATH='$${AXIOM}/bin':'$${PATH}' >>${COMMAND}
@echo export PATH >>${COMMAND}
@cat ${INT}/sman/axiom >>${COMMAND}
@chmod +x ${COMMAND}
@echo 79 Axiom installation finished.
@echo
@echo Please add $(shell dirname ${COMMAND}) to your PATH variable
@echo Start Axiom with the command $(shell basename ${COMMAND})
@echo
```

<div align="center">———-</div>

3.8　document

Each file in the system is in pamphlet form. This stanza, which is not executed by default, will walk the directories generating the documentation for each file. These are dvi files and since they are system-independent and machine-generated they live in the INT subdirectory. In particular, we will build an INT/DOC subtree mirroring the LSP and SRC subtrees. The INT/DOC subtree can be removed with no ill effect. Since all of the pamphlet files live in either the LSP or SRC subdirectories we make sure the INT/DOC/LSP and INT/DOC/SRC directories exist.

— **document** —

```
document:
@echo 22 documenting files
@(cd lsp ; ${ENV} ${MAKE} document )
@(cd src ; ${ENV} ${MAKE} document )
```

————-

3.9　clean

— **clean** —

```
clean:
@ echo 7 making a ${SYS} system, PART=${PART} SUBPART=${SUBPART}
@ echo 8 Environment ${ENV}
@ rm -f lsp/Makefile.dvi
@ rm -f lsp/Makefile
@ rm -f trace
@ rm -f *~
@ rm -f Makefile.${SYS}
@ rm -rf ${MNT}
@ rm -rf int
@ rm -rf obj
@ rm -rf mnt
@ for i in `find src -name "Makefile"` ; do rm -f $i ; done
@ for i in `find src -name "Makefile.dvi"` ; do rm -f $i ; done
```

————-

4　The Platform Makefiles

The Top Level Makefile examines the SPAD variable to determine the target build platform. It sets up the general structure of the world. Then it invokes one of these platform Makefiles. Each of these Makefiles sets several environment variables that are specific to this platform.

4.0.1 The LDF variable

The LDF variable is the generic loader flags. This gives information about where various libraries are located on specific platforms. On linux, for instance, the library libXpm.a is used by the graphics routines. This library is usually found in /usr/X11R6/bin/libXpm.a. Thus, on the linux platform LDF is defined as

```
LDF=" -L/usr/X11R6/lib -L/usr/lib ${XLIB}/libXpm.a -lXpm"
```

4.0.2 The AWK variable

On most systems the gnu toolset is the default. Thus we can just use 'awk' and the program works. However, on some systems we need to specify that we are using the gnu toolset, and we need to use gawk instead of awk.

4.0.3 The PATCH variable

On most systems the gnu toolset is the default. Thus we can just use 'patch' and the program works. However, on some systems we need to specify that we are using the gnu toolset, and we need to use gpatch instead of patch.

4.0.4 The O variable

Various Common Lisp systems prefer certain filename extensions. This defaults to "o" so a compile of foo.lisp becomes foo.o but other systems prefer .fasl so a compile of foo.lisp becomes foo.fasl. Change this based on the target lisp.

4.0.5 The LISP variable

There are 3 kinds of "lisp" files in the Axiom build process. The first are the "clisp" files. These are common lisp files generated by the boot compiler. The second are the "lisp" files. These are hand written common lisp code. The third are the "lsp" files. These are files generated by GNU Common Lisp. The LISP variable is used to name the third kind of files as this may change when the underlying common lisp is changed.

4.0.6 The DAASE variable

Axiom uses 5 files, the *.daase files, which are called "the databases". They contain cross-reference, signatures, and other information needed by the interpreter and compiler. When the system is being built from scratch these databases need to exist. However, they get dynamically rebuilt after the algebra files are compiled. The bootstrap versions of these databases live in the src/share subdirectory. Axiom will use the value of the shell variable DAASE to find its databases. If this variable is unbound it uses the standard `${MNT}/${SYS}` path. Note that Axiom will append the string /algebra to the value of DAASE. The default value setting given here is:

```
DAASE=${SRC}/share
```
so `${SRC}/share/algebra/*.daase` will be the Axiom bootstrap database files.

4.0.7 The XLIB variable

The XLIB variable tells us where the X11 libraries live. Axiom needs to use libXpm.a to build the graph subdirectory.

4.0.8 The SRCDIRS variable

The SRCDIRS variable is used in the src/Makefile.pamphlet to decide what directories to build on a given platform. This is needed at the moment because certain functions do not yet work on all platforms.

4.0.9 The GCLVERSION variable

GCLVERSION is the name of the GCL version. The one we used to build the original version of the system is gcl-2.4.1. The system will attempt to untar a file in the ZIPS directory with the name GCLVERSION.tgz, cd to the GCLVERSION subdirectory and do a ./configure followed by a make.

The GCLVERSION variable is also used to make the GCLDIR variable. GCLDIR tells depsys where GCL lives. The depsys image needs to load a file from GCL (cmpnew/collectfn.lsp) which is used to generate optimizations for function calling in Axiom. This is handled automatically by changing this variable.

If GCLVERSION is "gcl-system", then GCL is not built locally, and it is assumed that the "gcl" command is available off the path. IF this GCL is unsuitable for building Axiom then very bad things will happen.

NOTE WELL: IF YOU CHANGE THIS YOU SHOULD ERASE THE lsp/Makefile FILE. This will cause the build to remake the lsp/Makefile from the lsp/Makefile.pamphlet file and get the correct version. If you forget to erase the lsp/Makefile the wrong patches will be applied.

— GCLVERSION —

```
#GCLVERSION=gcl-2.4.1
#GCLVERSION=gcl-2.5
#GCLVERSION=gcl-2.5.2
#GCLVERSION=gcl-2.6.1
#GCLVERSION=gcl-2.6.2
#GCLVERSION=gcl-2.6.2a
#GCLVERSION=gcl-2.6.3
#GCLVERSION=gcl-2.6.5
#GCLVERSION=gcl-2.6.6
#GCLVERSION=gcl-2.6.7pre
#GCLVERSION=gcl-2.6.7
#GCLVERSION=gcl-2.6.8pre
#GCLVERSION=gcl-2.6.8pre2
#GCLVERSION=gcl-2.6.8pre3
#GCLVERSION=gcl-2.6.8pre4
#GCLVERSION=gcl-2.6.8pre7
```

```
#GCLVERSION=gcl-cygwin
#GCLVERSION=gcl-2.6.9
#GCLVERSION=gcl-2.6.10
#GCLVERSION=gcl-2.6.11
GCLVERSION=gcl-2.6.12
```

————-

4.0.10   The GCLOPTS configure variable

The GCLOPTS lisp requires some parameters for building which vary from system to system. We create an environment variable here so we can add options to the configure command in the lsp/Makefile.pamphlet.

— **GCLOPTS** —

```
GCLOPTS="--enable-vssize=65536*2 --disable-xgcl --disable-tkconfig"
```

————-

It turns out that we can successfully build GCL on many more systems if we set the GCLOPTS to build a local BFD.

We are failing during build because "directoryp is undefined" along with the message

```
Error: Cannot get relocated section contents
```

— **GCLOPTS-LOCBFD** —

```
GCLOPTS="--enable-vssize=65536*2 --disable-xgcl --disable-tkconfig"
```

————-

For the gcl-2.6.8pre7 version we move to using the custreloc option.

— **GCLOPTS-CUSTRELOC** —

```
GCLOPTS="--enable-vssize=65536*2 --disable-xgcl --disable-tkconfig"
```

————-

For the MACOSX port we need the following options. The "–disable-nls" means that we will not be supporting natural language internationalization. The "–enable-maxpage" has been eliminated because it causes build failures. The "–enable-machine" parameter appears to be used by configure from the setting of the "canonical" variable, which is in turn set by a shell script. We need to add "–enable-locbfd" and "–disable-dlopen" due to the error "unexec: not enough room for load commands for new __DATA segments".

— **GCLOPTS-MACPORT** —

```
GCLOPTS="--enable-vssize=65536*2 --disable-xgcl --disable-tkconfig"
```

————-

4.1 Makefile.freebsd

— Makefile.freebsd —

```
\getchunk{ENVDEFAULTS}
PLF=BSDplatform
CCF="-O2 -pipe -fno-strength-reduce -Wall -D_GNU_SOURCE \
      -D${PLF} -I/usr/X11R6/include -I/usr/local/include"
LDF="-L/usr/X11R6/lib -L/usr/local/lib"
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.FreeBSD called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

4.2 Makefile.windows

This is for the Windows port. We assume that the build will be done using GCC under MSYS.

We've modified the GCLOPTS variable from the standard config in two ways. First we add –enable-debug so more information is available for testing and second we removed –enable-statsysbfd.

— Makefile.windows —

```
\getchunk{ENVDEFAULTS}
AWK=awk
PLF=MSYSplatform
CCF="-O2 -Wall -D_GNU_SOURCE -D${PLF}"
SRCDIRS="bootdir interpdir sharedir algebradir etcdir docdir inputdir"
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}
```

```
all: srcsetup lspdir srcdir
@echo 45 Makefile.windows called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
        then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

— **Makefile.slackware** —

```
\getchunk{ENVDEFAULTS}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.slackware called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
        then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.3   Makefile.linux

— **Makefile.linux** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}
```

```
all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

——————-

4.4   Makefile.mandriva

— Makefile.mandriva —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.mandriva called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

——————-

4.5   Makefile.vector

— Makefile.vector —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on 'date' | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
            then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.6 Makefile.redhat72

— **Makefile.redhat72** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on 'date' | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
            then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.7 Makefile.redhat9

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

4.8   Makefile.debian

— **Makefile.debian** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

4.9   Makefile.opensuse

<div align="center">— Makefile.opensuse —</div>

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

<div align="center">————-</div>

4.10   Makefile.ubuntu

<div align="center">— Makefile.ubuntu —</div>

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.11 Makefile.mint

— **Makefile.mint** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on ‘date‘ | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.12 Makefile.ubuntu64

— **Makefile.ubuntu64** —

```
\getchunk{ENVDEFAULTS}
XLIB=/usr/lib
LDF:=" -L/usr/X11R6/lib -L/usr/lib ${XLIB}/libXpm.a -lXpm"
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on ‘date‘ | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
```

```
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

4.13   Makefile.centos

— **Makefile.centos** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

4.14   Makefile.macosxppc

— **Makefile.macosxppc** —

```
\getchunk{ENVDEFAULTS}
PLF=MACOSXplatform
CCF="-O2 -fno-strength-reduce -Wall -D_GNU_SOURCE -D${PLF} -I/usr/X11/include"
\getchunk{GCLOPTS-MACPORT}
\getchunk{ENVAR}


all: srcsetup lspdir srcdir
@echo 45 Makefile.macosxppc called
```

```
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.15   Makefile.fedora

— Makefile.fedora —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}


all: srcsetup lspdir srcdir
@echo 45 Makefile.fedora called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.16   Makefile.fedora5

— Makefile.fedora5 —

```
\getchunk{Makefile.fedora}
```

———-

4.17 Makefile.fedora6

— **Makefile.fedora6** —
```
\getchunk{Makefile.fedora}
```
————-

4.18 Makefile.fedora7

— **Makefile.fedora7** —
```
\getchunk{Makefile.fedora}
```
————-

4.19 Makefile.fedora8

— **Makefile.fedora8** —
```
\getchunk{Makefile.fedora}
```
————-

4.20 Makefile.fedora8-64

Fedora 8 on a 64 bit machine requires gcl-2.6.8pre2

— **Makefile.fedora8-64** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}


all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
        then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

4.21 Makefile.fedora9

— Makefile.fedora9 —

```
\getchunk{Makefile.fedora}
```

————-

4.22 Makefile.fedora10

— Makefile.fedora10 —

```
\getchunk{Makefile.fedora}
```

————-

4.23 Makefile.fedora11

— Makefile.fedora11 —

```
\getchunk{Makefile.fedora}
```

————-

4.24 Makefile.fedora12

— Makefile.fedora12 —

```
\getchunk{Makefile.fedora}
```

————-

4.25 Makefile.fedora13

— Makefile.fedora13 —

```
\getchunk{Makefile.fedora}
```

————-

4.26 Makefile.gentoo

— **Makefile.gentoo** —

```
\getchunk{ENVDEFAULTS}
\getchunk{ENVAR}


all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.27 Makefile.fedora64

— **Makefile.fedora64** —

```
\getchunk{ENVDEFAULTS}
LDF="-L/usr/X11R6/lib64 -L/usr/local/lib64 -L/usr/openwin/lib64 -L/usr/lib64 "
\getchunk{GCLOPTS-CUSTRELOC}
\getchunk{ENVAR}


all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
          then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
```

```
\getchunk{clean}
```

———-

4.28   Makefile.fedora3

Fedora Core 3 needs special GCL options for local bfd.

— **Makefile.fedora3** —

```
\getchunk{ENVDEFAULTS}
\getchunk{GCLOPTS-LOCBFD}
\getchunk{ENVAR}


all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on 'date' | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
           then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

———-

4.29   Makefile.MACOSX

On the MAC OSX someone decided (probably a BSDism) to rename the SIG-
CLD signal to SIGCHLD. In order to handle this in the low level C socket code
(in particular, in `src/lib/fnct_key.c`) we change the platform variable to be
MACOSXplatform and create this new stanza.

It appears that the MAC OSX does not include gawk or nawk by default so
we change the AWK variable to use awk.

We need to add -I/usr/include/sys because `malloc.h` has been moved on
this platform.

We need to search includes ("-I") in /usr/include before /usr/include/sys
because the MAC seems to search in a different order than linux systems. The
sys versions of the include files are broken, at least for Axiom use.

— **Makefile.MACOSX** —

```
\getchunk{ENVDEFAULTS}
AWK=awk
PLF=MACOSXplatform
CCF="-O2 -fno-strength-reduce -Wall -D_GNU_SOURCE -D${PLF} \
     -I/usr/X11/include -I/usr/include -I/usr/include/sys"
\getchunk{GCLOPTS-MACPORT}
\getchunk{ENVAR}

all: srcsetup lspdir srcdir
@echo 45 Makefile.linux called
@echo 46 Environment : ${ENV}
@echo 47 finished system build on `date` | tee >lastBuildDate
@- if [ "${TESTSET}" != "notests" ] ; \
            then  grep "result FAILED" int/input/*.regress || true ; fi

\getchunk{literate commands}
\getchunk{srcsetup}
\getchunk{src}
\getchunk{lsp}
\getchunk{document}
\getchunk{clean}
```

————-

References

[1] CMUCL **http://www.cons.org/cmucl**

[2] GCL **http://savannah.gnu.org/projects/gcl**

[3] Codemist Ltd, "Alta", Horsecombe Vale Combs Down Bath BA2 5QR UK Tel. +44-1225-837430 **http://www.codemist.co.uk**

[4] $SPAD/zips/advi-1.2.0.tar.gz, the advi source tree