NMG Documentation

Part III. Maintainer's Guide

RECEIVED

OCT 24 1996

OSTI

Frederick N. Fritsch
Robert P. Dickinson, Jr.

July 1996

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

NMG Documentation

Part III. Maintainer's Guide

Frederick N. Fritsch
Robert P. Dickinson, Jr.

Mathematical Software Service
Livermore Computing Department
Lawrence Livermore National Laboratory

This is the third of a three-part report documenting NMG, the Numerical Mathematics Guide. Part I is aimed at the user of the system. It contains an introduction, with an outline of the complete report, and Chapter 1, User's Point of View. Part II is aimed at the programmer and contains Chapter 2, How It Works. Part III is aimed at the maintainer of NMG and contains Chapter 3, Maintenance, and Chapter 4, Validation. Because its contents are so specialized, Part III will receive only limited distribution. Note that each chapter has its own page numbering and table of contents.

DISCLAIMER

**Portions of this document may be illegible
in electronic image products. Images are
produced from the best available original
document.**

Chapter 3. Maintenance

3.0. Overview

This chapter describes the tools and procedures that are used to maintain the various data files required to support the NMG system, as well as the archival structure used for back-up.

Following is an overview of the contents of the remainder of this chapter. We discuss the procedures and tools used to generate and/or update various databases and files required to support NMG functions. With that background, we are able to describe how to add a whole new library to NMG or how to add or delete routines from an existing library. We conclude with a discussion of our back-up procedures and archive directory structure.

Contents

3.0. Overview	1
3.1. Helpfile Maintenance	4
3.1.0. Basic Information	4
3.1.1. Maintenance Procedure	4
3.1.2. Tool Abstracts	6
3.1.3. Constructing HELPCHK	7
3.1.4. Helpfile Revision Dates	9
3.2. Advisor Database Maintenance	10
3.2.0. Basic Information	10
3.2.1. Repairing Decision Trees Using the Label Last Form	10
3.2.2. Producing an Updated Database from llast1	12
3.2.3. More Details	15
3.3. MenuDriver Database Maintenance	17
3.3.0. Basic Information	17
3.3.1. Deletion	17
3.3.1.1. Tool Descriptions	18
3.3.1.2. File Formats	20
3.3.1.3. A Example	21
3.3.2. Reclassification	23
3.3.2.1. Tool Descriptions	23
3.3.2.2. Input File Formats	26
3.3.2.3. A Example	26
3.3.3. Insertion	29
3.3.3.1. Tool Descriptions	29
3.3.3.2. Input File Formats	30
3.3.3.3. A Example	31

3.3.4. Menu Fixing.....	32
3.3.5. Comparison	32
3.3.6. Other Tools	33
3.4. Extracting Information from SLATEC-Format Source Codes.....	35
3.4.1. Use of EXDOC to Extract On-Line Documentation.....	35
3.4.2. The runexdoc Script.....	39
3.4.3. Use of EXINFO to Extract Purposes and GAMS Codes	40
3.4.4. The runexinfo Script	41
3.5. Superfile Generation	43
3.5.1. Hand-generated Superfiles	43
3.5.2. Generating Superfiles for SLATEC-format Routines	43
3.5.3. Using sfg.....	44
3.6. How to Install a Library in NMG	45
3.6.0. Installation Overview.....	45
3.6.1. Documentation	45
3.6.2. Source Code.....	46
3.6.3. Superfiles.....	46
3.6.4. Legal Notices.....	47
3.6.5. MenuDriver Databases	47
3.6.6. Advisor Databases	47
3.6.7. Library Validation	48
3.6.8. Helpfile Updates	48
3.6.9. User Notification	48
3.6. Appendix. Library- ~~fetch~~ -file Correspondence Table.....	48
3.7. How to Add or Delete a Routine in a Library	49
3.7.1. Adding a Routine	49
3.7.2. Deleting a Routine	50
3.7.3. Replacing a Routine	50
3.8. Processing NMG Usage Statistics	52
3.8.1. How Statistics are Generated.....	52
3.8.2. Monthly Statistics.....	53
3.8.3. Quarterly Statistics.....	54
3.8.4. Tool Abstracts.....	56
3.9. Back-up Procedures	57
3.9.1. File Naming Conventions	57
3.9.2. Cray Directory Structure	57
3.9.3. SCF Archive Directory Structure	57
3.9.4. Advisor and MenuDriver Databases	59
3.9.5. Helpfiles	59
3.9.6. Fetch Back-up Files	59

3.9.7. GAMS Files	59
3.9.8. Where Are the Tools?.....	60
3.9.9. Usage Statistics.....	60
3.10. Notes on Future Development.....	61
3.10.1. What Kinds of Updates Need to Be Performed?	61
3.10.2. What Other Tools Would Be Helpful?	61
3.Appendix. Tool Directory.....	63

3.1. Helpfile Maintenance

This section describes the tools and procedures used to maintain the helpfiles used by the NMG Help Facility. Refer to Section 2.3 for the formats of helpfiles and the associated list-files, as well as listings of the current list-files.

3.1.0. Basic Information

In order to achieve our maintenance goals, help screens are kept as individual files in an ar-format archive. In addition to the list-files and help screen files (helpfiles), the help screen archive also contains scripts used in the maintenance of these archives. (See its README file.) These are kept in directory `.m29/.nmg/unicos/help/screens` (SCF archive).

In order to simplify the previewing of newly-updated help screens, a special version of the Help Processor, called `HELPCHK`, has been created in our UNICOS working directory `/usr/local/math/nmg/help/work`. This assumes the standard list-files and sub-directory helpfiles are in the current working directory and prompts the user for the name of a list-file. It then starts up with the first-listed helpfile and allows the user to view any of the other helpfiles via the "help" command. See Appendix A for more information on `HELPCHK`.

3.1.1. Maintenance Procedure

The following procedure is recommended for maintaining NMG help screens. Refer to the Tool Abstracts subsection, below, for detailed information on scripts and files used here.¹

1. Read the most recent help screen archive from SCF archive directory `.m29/.nmg/unicos/help/screens` to our UNICOS working directory `/usr/local/math/nmg/help/work`. (Archives have names of the form `hlpscrn.yymmdd`, to indicate the version updated on `yy-mm-dd`. For example, `hlpscrn.930209` was updated February 9, 1993.)²

NOTE: In the following, this is called `hlpscrn.old`.

- 1a. Clean-up becomes easier if one first creates a work directory in the UNICOS working directory:

```
mkdir mytemp
cp hlpscrn.yymmdd mytemp/hlpscrn.old
cd mytemp
```

2. Extract contents of help screen archive and save a copy of current help screens:

```
ar xv hlpscrn.old
mkdir hsf.old
copyem lsthlpscrns hsf.old
```

¹ In order of appearance: `copyem`, `lsthlpscrns`, `comparem`, `catem`, `comparcp`, `movem`, `updhshf`.

² Check: This file may already be in `/usr/local/math/nmg/help/work`.

3. (Optional) Verify that these correspond to the floor versions of the help files by using `comparem`:

```
comparem ../helpfiles hsf.old
```

(If this has been done in a subdirectory, change `".."` in this command to `"../.."` to move up two levels to `/usr/local/math/nmg/help`.) The summary at the end should show no `diff` errors if these are the current helpfiles.³

4. Update individual help screens as appropriate, using your favorite editor. *Be sure to update the revision date (and initials, if appropriate) on the title line of any modified help screen file!*

- a. Check revisions:

```
mkdir hsf.new <--- Not needed if iterating.
```

```
copyem lsthlpscrns hsf.new
```

```
comparem hsf.old hsf.new > comp.out
```

Review `comp.out` to *make sure that all updated screens have updated dates on title line, and that desired changes have been made.*

- b. If revisions are extensive, you may wish to

```
catem lsthlpscrns > cathlpscrns
```

and print file `cathlpscrns` for proofreading off-line. Iterate this step as often as necessary.

5. Run `HELPCHK` on any modified help files to make sure that each screen performs as expected. (Return to Step 4, if necessary, to correct problems.)

6. Update the floor versions as necessary:

```
comparcp hsf.new ../helpfiles
```

(As before, change `".."` to `"../.."` if this is run in a subdirectory.) Run `nmg` to exercise the updated help files, as a double-check.

7. Update the help screen archive:

```
movem lsthlpscrns hsf.new <--- To make sure that final versions are in
                                hsf.new.
```

```
comparem hsf.old hsf.new > comp.outck
```

```
diff comp.out comp.outck <--- To verify that the final versions are
                                what we last checked.
```

```
cp hlpscrn.old hlpscrn.yymmdd <--- Today is yy-mm-dd.
```

```
updhsh hlpscrn.yymmdd hsf.new >& updhsh.log
```

Check `updhsh.log` to make sure that all went as expected. Then update `README` with date and reason for update(s) and:

```
ar rv hlpscrn.yymmdd README
```

Finally, check against previous version:

```
ar tv hlpscrn.old > lstscrn.old <--- May already be here.
```

```
ar tv hlpscrn.yymmdd > lstscrn.yymmdd
```

³ Exception: The generated file `helpmenu`, if it exists on `hsf.old`, may be different from the one in the floor directory, since it is created anew each time the Help Facility is invoked.

```
diff lstscrn.old lstscrn.yymmdd > difhsfar
```

Review difhsfar to verify that all expected files (and no others) have been modified. If OK, save hlpscrn.yymmdd and lstscrn.yymmdd to archive directory .m29/.nmg/unicos/help/screens.

8. Clean up:

```
rm -r hsf.old hsf.new
```

Remove any other scripts and files left laying around that are no longer needed. Note: If above work was done in a temporary directory, as in 1a, above, this simplifies to:

```
cd ..
rm -r mytemp
```

3.1.2. Tool Abstracts

The following files may be found in .m29/.nmg/unicos/help/tools in the SCF archive. These are all C-shell scripts. Some of them are also included in the hlpscrn archive. More details on a script may usually be found in the comments at the beginning of the script file itself.

Program name: catem (concatenate them)

Purpose: Run cat all of the files in a list.

Usage: catem list > catfile

Input: list is a text file containing a list of file names, one per line.

Output: catfile differs from a simple cat of the files in that it includes a header with the name of the file before each of them.

Notes: For example, "catem lsthlpscrns > cathlpscrns" gives a listing of all help screens, which can be printed for proofreading.

Program name: comparcp (compare and copy)

Purpose: Compare each file in one directory against the file with the same name in another directory, copying any that differ from the first to the second.

Usage: comparcp drev dcomp > comp.log

Input: drev is a directory containing possibly revised helpfiles, and dcomp is the comparison directory to be updated.

Output: comp.log will be a logfile indicating which files have been copied.

Notes: Useful for updating the public helpfile directory after checking out a revised set of helpfiles.

Program name: comparem (compare them)

Purpose: Compare each file in one directory against the file with the same name in another directory.

Usage: comparem dbase dcomp > comp.out

Input: dbase is the first (usually the old or "base") directory, and dcomp is the second (or comparison) directory.

Output: comp.out will contain the output of the comparison (omit "> comp.out" to see output at the terminal).

Notes: This tool is useful for many other purposes, as well.

Program name: copyem (copy them)

Purpose: Copy files to a directory.

Usage: copyem list dir

Input: list is a file containing the names of files to be copied from current working directory, and

dir is the destination directory.

Output: dir will contain copies of the indicated files.

Notes: list is frequently lsthlpscrns. (See also movem.)

Program name: lsthlpscrns (list of help screens)

Purpose: File containing the names of the current helpfiles.

Notes: This is not a program, but rather a list file for use with catem, copyem, or movem.

Program name: movem (move them)

Purpose: Move files to a directory.

Usage: movem list dir

Input: list is a file containing the names of files to be moved from current working directory, and

dir is the destination directory.

Output: dir will contain the indicated files.

Notes: list is frequently lsthlpscrns. (See also copyem.)

Program name: updhfsf (update help screen file archive)

Purpose: Update a NMG help screen archive.

Usage: updhfsf hsfar dir > updhfsf.log

Input: hsfar is an NMG help screen archive (ar format), and

dir is a directory containing one or more (possibly updated) help screens.

Output: hsfar will be updated, and

updhfsf.log will be a logfile indicating which files have been updated.

Notes: The object is to replace only changed help screen files. Typically hsfar will be a copy of hlpscrn.old and dir will be a directory containing the newly updated and verified help screens.

3.1.3. Constructing HELPCHK

The HELPCHK program is constructed by replacing the main program of help.f with the following. (This is file mainchk.f in both /usr/local/math/nmg/help/work and in the SCF archive.⁴ The listing has had some comments deleted for inclusion in this report.)

---

⁴ This will replace mainh.f in the assembled source code. For example, do "cat mainchk.f help.f > helpchk.f", invoke an editor to delete the main code that came from help.f, and compile the result via "f77 -o helpchk helpchk.f".

```

*DECK MAINCH
      PROGRAM HELP
C***BEGIN PROLOGUE  MAINCH
C***PURPOSE  NMG Helpfile Checker Main Code
C***LIBRARY   NMG
C***TYPE      ALL (MAINCH-A)
C***AUTHOR  Dickinson, Robert P., Jr., (LLNL/LC/MSS)
C           Fritsch, Frederick N., (LLNL/LC/MSS)
C***DESCRIPTION
C
C   Usage:  helpchk
C   (code)   NMG Helpfile Checker
C   (code)
C   (code) Type name of listfile.
C   (user) listname
C
C   where listname is the name of the listfile to be used.
C           (Must be in current working directory.)
C
C***ROUTINES CALLED  HELPS2, LASTNB
C***REVISION HISTORY  (YMMDD)
C   900827  DATE WRITTEN
C   951228  Made special version to check out new screens. (FNF)
C***END PROLOGUE  MAINCH
      IMPLICIT NONE
C
C   Declare externals
C
C   INTEGER LASTNB
C   EXTERNAL HELPS2, LASTNB
C
C   Declare variables
C
C   INTEGER NCHDIR, NCHLST
C   CHARACTER*16 LSTNAM
C   CHARACTER*40 HLPDIR
C   CHARACTER*56 HFDIR, HFLST
C
C   Initialize
C
C***FIRST EXECUTABLE STATEMENT  MAINCH
      HLPDIR = '.'
      NCHDIR = LASTNB(HLPDIR)
      HFDIR = HLPDIR(1:NCHDIR) // '/helpfiles'
C
C   Issue greeting and show first screen
C
C   WRITE (*,*)
C   WRITE(*,2)
C   2 FORMAT(5X,'NMG Helpfile Checker')
C
C   3 CONTINUE
C   WRITE (*, '(A)') 'Type name of listfile.'
C   READ (*, '(A)') LSTNAM
C   NCHLST = LASTNB(LSTNAM)

```



```

C      Check whether user simply typed a carriage return.
      IF (NCHLST .LE. 0) GO TO 3
      HFLST = HLPDIR(1:NCHDIR) // '/' // LSTNAM(1:NCHLST)
C
      CALL HELPS2('1',HFLST,HFDIR)
      STOP
C ----- Last line of MAINCH follows -----
      END

```

Note that the way HELPS2 works (see Section 2.3.1), it is not possible for the same code to cycle for additional list-files. Instead, it must be restarted to try a different list-file.

3.1.4. Helpfile Revision Dates

The following output from "head -1 helpfiles/*" shows the revision date for each of the current help screens on the LC Cray J90's.

==> helpfiles/advmen <==	
Main Menu Commands	Revised: 93-07-12.
==> helpfiles/advov <==	
Advisor Overview	Revised: 93-07-13 (FNF).
==> helpfiles/advstd <==	
Standard Screen Commands	Revised: 93-07-12 (FNF).
==> helpfiles/fetch <==	
Fetch Overview	Revised: 91-05-10 or earlier.
==> helpfiles/helpov <==	
Help Overview	Revised: 94-05-24 (FNF).
==> helpfiles/libabb <==	
Library Abbreviations	Revised: 95-08-30 (FNF).
==> helpfiles/libava <==	
Library Availability	Revised: 96-01-12 (FNF).
==> helpfiles/mailov <==	
Mail Overview	Revised: 94-05-24 (FNF).
==> helpfiles/mdrcat <==	
Category Search Commands	Revised: 93-07-12 (FNF).
==> helpfiles/mdrdsc <==	
Description Format Notes	Revised: 95-12-04 (FNF).
==> helpfiles/mdrgen <==	
General Information	Revised: 93-07-12 (FNF).
==> helpfiles/mdrlib <==	
Library Selection Display	Revised: 93-07-13 (FNF).
==> helpfiles/mdrlst <==	
List Continuation Commands	Revised: 94-06-02 (FNF).
==> helpfiles/mdrov <==	
Menu Overview	Revised: 93-07-12 (FNF).
==> helpfiles/nmgov <==	
NMG Overview	Revised: 95-12-28 (FNF).

3.2. Advisor Database Maintenance

This section describes the tools and procedures used to update Advisor databases. These same tools are also appropriate for the first, or "menu" portion of a MenuDriver database. It is assumed that the reader is familiar with the database formats described in Sections 2.4–2.5.

3.2.0. Basic Information

Repairing the menu portion for Advisor and MenuDriver databases involves the use of two C Shell scripts, `menu1` and `menu2`. `menu1` takes as input a database and creates a text file called `llast1` containing the menu portion in an editable format.¹ The user repairs the text file `llast1` using an editor such as `vi`. After repairs are complete on `llast1`, the user runs the second script, `menu2`, which produces as output a text file called `newtree`. If done correctly, `newtree` is the new repaired database.

In addition to producing a new database, `menu2` destroys temporary files produced during the repair process. It does not destroy the original database. Thus using a utility such as `diff` one may compare the new database with the old one. Alternatively, the Fortran program `COMP5` can be run to compare the menu portions of two databases node by node and report discrepancies (see description at end of section).

3.2.1. Repairing Decision Trees Using the Label Last Form

If we use the database `intree3` listed in Section 2.4.2 as input to script `menu1`, the `STIME1` code which it invokes produces the following output file `llast1`:

```

adrive 02/09/96 FNF
Solutions of Nonlinear Equations                      3
Single equation in one unknown                        3a
All roots of a polynomial equation                    3a1
  Use RPZERO (SLATEC)
General equation                                      3a2
An interval in which the desired root lies is known   3a2a
  Use ZEROIN (MATHLIB) or
  Use SZERO (PMATH).
Although no such interval is known, some estimate of  3a2b
the root is known
  Use C05AJF (NAG)
System of equations                                   3b
  Use SNSQE (SLATEC)
An alternate way of solving a system of the form      i
  f1(x) = 0, f2(x) = 0, ... (n equations in the n-vector x) i
is to find a minimum of the sum of squares           i
  [f1(x)]**2 + [f2(x)]**2 + ...                      i
Consult the interpolation/approximation tree for such routines. i

```

¹ It also writes the `MTABLE` part of a MenuDriver database into file `tail1`.

Note that the nodes are listed in the same order that a user would see them when running Advisor, if the tree were fully expanded, but with the labels printed last (hence the term "label last form").

The general format of a `llast1` file is:

- The first line contains a character string (A):
 TITLE This CHARACTER*40 string begins with the word `adrive` or `mdrive`, to indicate an Advisor or MenuDriver database. This is followed by identification information.. (This is simply a copy of columns 36–75 of the first line of the input database.)
- The remaining lines are character strings, read into a CHARACTER*100 array. (The file terminates with an end-of-file.)

The lines are grouped into blocks, where each block contains the text associated with a particular node. At the end of the first line of each block is a label, which is the name associated with that node; call this line the *label line*. The label begins in column 71 of the line; it is a character string of length 24, the maximum size for a label. One easily sees the blocks, for each ends at the line before another label line. These blocks are of two types: ordinary blocks and information blocks.

Ordinary blocks: An ordinary block starts with a label line. A label line consists of a text portion followed by the label, as described above. The text portion lies in columns 1 through 64. An ordinary block may consist of one or more lines. All lines beyond the first line are text lines and again occupy columns 1–64; column 65 and beyond should be blank.²

Information blocks: More information is supplied at certain nodes of the database. These nodes have information blocks in addition to ordinary blocks. The first part of the text for an information node is in the same format as an ordinary block, and is called the *ordinary part* of the information node. The lines following this portion contain text in columns 1 through 70, followed by the character "i" (must be lower case) in column 71. This part is called the *information block* of the information node.³

When a user first sees a data tree, only ordinary blocks are displayed by Advisor or MenuDriver. The user also sees the character "i" displayed on the label line of information nodes. If he/she types "`i lab`", the information block at node `lab` becomes visible. Information blocks may be long, which is the main reason for hiding them initially.

Note that the labels in label lines must be given in depth-first (lexicographic) order. Thus, if we wished to subdivide node 3a1, its children would be inserted before the text for node 3a2. Thus, we might edit the above to produce the following updated `llast1` file. (Refer to the Advisor database description in Section 2.4.1 for details on the allowable formats for Use-blocks in recommendations.)

---

² Advisor and MenuDriver display only columns 1–64 of ordinary blocks.

³ Advisor and MenuDriver display columns 1–70 of information blocks.

```

adrive 02/09/96 FNF
Solutions of Nonlinear Equations          3
Single equation in one unknown             3a
All roots of a polynomial equation         3a1
Real coefficients                          3a1a
    Use RPZERO (SLATEC)
Complex coefficients                       3a1b
    Use CPZERO (SLATEC)
General equation                           3a2
An interval in which the desired root lies is known 3a2a
    Use ZEROIN (MATHLIB) or
    Use SZERO (PMATH).
Although no such interval is known, some estimate of 3a2b
the root is known
    Use C05AJF (NAG)
System of equations                        3b
    Use SNSQE (SLATEC)
An alternate way of solving a system of the form      i
    f1(x) = 0, f2(x) = 0, ... (n equations in the n-vector x) i
is to find a minimum of the sum of squares           i
    [f1(x)]**2 + [f2(x)]**2 + ... .                  i
Consult the interpolation/approximation tree for such routines. i

```

Note that it is not necessary to update the date on the first line (although you may if you wish to), because it will be overwritten by the current date in the final database.

3.2.2. Producing an Updated Database from 1last1

Using file 1last1 as input, menu2 first runs the FRED1 code to produce an intermediate inrep file. It then runs REF1 to convert this to headm3, the updated menu. As a final step it concatenates headm3 with tail1 (which is empty for an Advisor database) to produce newtree, the final repaired database.

The inrep file. Following is the inrep file produced from the edited 1last1 for our example. Note that FRED1 has reordered the text from 1last1 into the width-first order that will appear in the final database.

```

adrive 02/09/96 FNF
0
21
Solutions of Nonlinear Equations          3
Single equation in one unknown             3a
System of equations                        3b
    Use SNSQE (SLATEC)
An alternate way of solving a system of the form      i
    f1(x) = 0, f2(x) = 0, ... (n equations in the n-vector x) i
is to find a minimum of the sum of squares           i
    [f1(x)]**2 + [f2(x)]**2 + ... .                  i
Consult the interpolation/approximation tree for such routines. i
All roots of a polynomial equation         3a1
General equation                           3a2
Real coefficients                          3a1a
    Use RPZERO (SLATEC)
Complex coefficients                       3a1b
    Use CPZERO (SLATEC)

```

An interval in which the desired root lies is known	3a2a
Use ZEROIN (MATHLIB) or	
Use SZERO (PMATH).	
Although no such interval is known, some estimate of	3a2b
the root is known	
Use C05AJF (NAG)	
9	
1	
3 2	
1 1 3	
2 1 3a	
3 7 3b	
4 1 3a1	
5 1 3a2	
6 2 3a1a	
7 2 3a1b	
8 3 3a2a	
9 3 3a2b	
1 2 3	3a
1 3 3	3b
2 4 3a	3a1
2 5 3a	3a2
4 6 3a1	3a1a
4 7 3a1	3a1b
5 8 3a2	3a2a
5 9 3a2	3a2b

The `inrep` file could be considered a "fundamental form" for Advisor trees. It is very useful for repairing trees.⁴ Its format is as follows:

- The first line contains a character string (A):
TITLE This CHARACTER*40 string begins with the word `adrive` or `mdrive`, to indicate an Advisor or MenuDriver database. This is followed by identification information. (This is simply a copy of the first line of the `llast1` file.)
- The second line contains a single number (I4):
NELIM number of nodes to be eliminated. (It is written as zero by `FRED1`.)
- If **NELIM** is positive, the next line(s) contain a list of the numbers of the **NELIM** nodes to be eliminated (10I4).
- The next line contains a single number (I4):
NL number of lines of text that contain the information about the nodes.
- The next **NL** lines contain the text from `llast1`, presented width-first as opposed to depth-first order.
- The next line contains a single number (I4):
ND number of nodes in tree.
- The next line contains a single number (I6):
NLMD number of nodes that are information nodes.

---

⁴ Before the `llast1` form was developed, the `inrep` file was updated directly by database maintainers.

- If NLMD is positive, the next NLMD lines contain LMD1, LMD2 pairs (2I4), as in the Advisor database format described in Chapter 2.⁵
 LMD1 node number of this information node.
 LMD2 number of lines at this node which are *not* part of the information block (do not have "i" in column 71).
- The next ND lines is a table of node numbers (consecutive integers), lines of text and node labels (2I4,2X,A):
 NDNUM node number.
 NTXT number of lines of text for node NDNUM (including any information block).
 [Note that these should sum to NL.]
 LAB node label (a CHARACTER*24 variable).
- The remainder of the file contains ND-1 pairs of node numbers indicating the node connectivities and corresponding node labels (2I4,2X,2A):⁶
 NPAR node number of parent.
 NCHI node number of child.
 LABELP parent label (a CHARACTER*24 variable).⁷
 LABELC child label (a CHARACTER*24 variable).

Producing a new database. Code REF1 reads the inrep file and converts it into an updated menu file, headm3. menu2 then catenates tail1 onto it to produce the final result, newtree.⁸

Although REF1 has coding for the NELIM option, this is a feature from an earlier version of the code. The present implementation does eliminate the indicated nodes (and all of their children). However, it re-labels nodes in the node-label table, but not in the text portion of the output file. (This would need to be corrected to make this a useful option.)

Here is the difference between the final result and the original database for our example:

```
milstein% diff intree3 newtree
1,4c1,5
<      7      10      18      0      1      adrive 02/09/96 FNF
<      2       3       4       5      0       0       6       7       0       0
<      1       3       5       6       7       9      10
<      1       2       3      10      12      13      16
---
>      9      13      21      0      1      adrive 02/22/96 rpd
>      2       3       4       5      0       6       7       8       9       0
>      0       0       0
>      1       3       5       6       8      10      11      12      13
>      1       2       3      10      11      12      14      16      19
15d15
< Use RPZERO (SLATEC)
```

⁵ These are generated from lines that have "i" in column 1 in the llast1 file.

⁶ Every node except the root is the child of exactly one other node, hence the total ND-1 here.

⁷ Note: LABELP and LABELC are stored in the indexed array called LAB.

⁸ This is a meaningless operation for an Advisor database. This feature is included to allow updating the menu part of a MenuDriver database.

```

16a17,20
> Real coefficients
3a1a
> Use RPZERO (SLATEC)
> Complex coefficients
3a1b
> Use CPZERO (SLATEC)
29,30c33,36
< 6 3a2a
< 7 3a2b
---
> 6 3a1a
> 7 3a1b
> 8 3a2a
> 9 3a2b

```

Note that there are two more nodes than before, and the IC, ICP, IPP arrays have been updated accordingly. Old nodes 6 and 7 have been renumbered 8 and 9 to make room for the two new ones. Note also that the date on the first line has been updated.

3.2.3. More Details

menu1 and menu2 invoke several Fortran programs to accomplish their tasks. These programs are stored in an ar-file called `tem41.a`. menu1 uses program STIME1 (show time); menu2 uses programs FRED1 and REF1 (reform the database). These programs are described below. menu2 concatenates files `headm3` (output from REF1) and `tail1` (output from STIME1) into file `newtree`. It then removes temporary files generated in making the new database.

In all of the following cases, the executable *NAME* is created as follows:⁹

```
f77 -o name name.f
```

Program name: STIME1 (show time)

Purpose: Split a MenuDriver database into its two parts. (Used by menu1.)

Type: Fortran program [source code: `stime1.f`]

Input: Advisor or MenuDriver database

Output: `llast1` and `tail1`

`llast1` contains the menu portion of the database in "label last" format, as described above.

`tail1` contains the part of the database following the menu portion. This file is empty if the database is type Advisor, otherwise it starts with MTABLE, a table linking MenuDriver text with menu tree labels. (See MenuDriver database description in Section 2.5.1.)

Notes: After `llast1` has been repaired, menu2 is invoked. It concatenates `tail1` onto the end of the new menu and then destroys `tail1`.

---

⁹ The standard Fortran compiler may be different on different machines. For example, one should use `cf77` instead of `f77` on the Cray; on an RS-6000 with AIX, use `xlf`.

Program name: FRED1

Purpose: Manipulate `llast1` into a data representation suitable for input into program REF1. (Used by menu2.)

Type: Fortran program [source code: `fred1.f`]

Input: `llast1`

Output: `inrep` (Format described above.)

Program name: REF1 (reform the database)

Purpose: Produce the menu portion of the new database. (Used by menu2.)

Type: Fortran program [source code: `ref1.f`]

Input: `inrep`

Output: `headm3` (header or menu portion of database)

Notes: menu2 concatenates files `headm3` and `tail1` into file `newtree`.

Following are descriptions of some miscellaneous tools that are useful for Advisor database maintenance.

Program name: LINECT (line count)

Purpose: Eliminate trailing blanks and report any text lines of length greater than 64 unless the line is part of an information block.

Type: Fortran program [source code: `linect.f`]

Input: `llast1`

Output: `llast1n` (new `llast1`)

A file where trailing blanks have been eliminated.

(Note that reports of long lines appear at the user's terminal.)

Notes: LINECT should be run after updating a `llast1` file, to eliminate extraneous blanks and make sure that no information in the ordinary part of updated blocks will be lost when displayed by Advisor.

Program name: COMP5

Purpose: Compare two databases node by node and report discrepancies.

Type: Fortran program [source code: `comp5.f`]

Input: Two databases in `llast1` format (COMP5 prompts user for their names).

Output: `compout`

Notes: A script `compm` has been written, which exercises COMP5 for the user. It prompts the user for the names of two databases (may be either Advisor or MenuDriver), runs STIME1 on each to get their `llast1` files, compiles and runs COMP5. The output first shows nodes that exist in both but differ, then it displays nodes that exist in only one of the databases.

3.3. MenuDriver Database Maintenance

This section describes the tools and procedures used to update MenuDriver databases. To update the first, or "menu" portion of a MenuDriver database, refer to Section 3.2. It is assumed that the reader is familiar with the database formats described in Sections 2.4–2.5.

3.3.0. Basic Information

MenuDriver databases may be thought of as divided into three parts, the header or menu portion, MTABLE, and descriptions of programs residing in the NMG collection of libraries.¹ MTABLE is a table of line numbers used for printing various descriptions as requested by a user. Repair of MenuDriver databases is accomplished using five functions: deletion, reclassification, insertion, menu fixing, and comparison. These functions are accomplished by the scripts and Fortran codes described below.

Historically, these five functions were used on a single database. Recently more scripts have been written which operate on all the databases at once or on a subset of them. These scripts were used most recently when a new version of the NAG library was installed. A mass delete was first performed (all databases at once); this was followed by a mass reclassification (the new NAG library was reclassified to fit into existing databases); this was followed by a mass insertion. In what follows below both types of scripts will be described. The new scripts only apply to deletion, reclassification, and insertion.

Following the descriptions of the five basic functions appears a set of brief descriptions of other tools useful in MenuDriver database maintenance. These include some of the codes that are invoked by the maintenance scripts. The description of Fortran program *NAME* is written as though the executable is created as follows:²

```
f77 -o name name.f
```

even though most scripts omit the `-o` field and simply execute the default `a.out`.

3.3.1. Deletion

The deletion process involves three C shell scripts and several Fortran programs used by these scripts. The scripts are called `lib1s` (library strip version 1), `strp1` (strip 1), and `strp2`. The purpose of these scripts is to delete (strip) designated descriptions from a collection of MenuDriver databases. The collection is a subset of all databases and may be of size one. `strp2` is a sub-script of `lib1s`, so there are really only two scripts to consider in detail.

These scripts and related files are stored in ar-file `strip.a`. The version last updated on *yy-mm-dd* is saved to SCF archive directory `.m29/.nmg/unicos/mdriver/tools` as `strip.yymmdd`.

---

¹ These last two parts, as a unit, were referred to as the MTABLE part of the database in Section 2.5.1.

² The standard Fortran compiler may be different on different machines. For example, one should use `cf77` instead of `f77` on the Cray; on an RS-6000 with AIX, use `xlf`.

3.3.1.1. Tool Descriptions. `lib1s` strips a collection of databases as specified by an input file called `cntrl` (control); this input file contains on each line the name of a routine, the name of the library, and the GAMS label. An example of a control file and how to generate it will be shown below. When `lib1s` is executed new databases are created, which are the old databases with descriptions deleted (or stripped) as specified by `cntrl`. Stripped descriptions are also saved. For example, if the old database is `intreeA`, then two files are produced, `ntreeA` (the new database), and `int2A` (discarded purposes, including headers if an entire node was collapsed). Normally, a file such as `int2A` may be removed, but it can also be edited and used for insertion back into the new database if desired. If one desires to reinsert `int2A`, for example, missing headers must be built in using some editor.

The difference between `lib1s` and `stripl` is that `stripl` only works on a single database instead of a collection. It prepares a file `ingam` of stripped descriptions as specified by `cntrl`. Duplicate descriptions have been removed as well as headers. `ingam` may be used as an input file for the reclassification function.

Program name: `lib1s` (library strip version 1)

Purpose: Delete descriptions from a collection of MenuDriver databases and make new databases

Type: C shell script

Usage: `lib1s`

Input: Collection of MenuDriver databases in the directory in which `lib1s` is run. These must have names of the form `intree $\alpha$` , where α is a capital letter.

Output: Collection of new MenuDriver databases and files of stripped descriptions. The outputs for `intree $\alpha$` will be in `ntree $\alpha$` and `int2 $\alpha$` .

Notes: `cntrl` contains information to strip certain descriptions; it must be present for `lib1s` to work. `cntrl` may be created by the user or may be constructed with the help of program `CNTRL`. `lib1s` invokes the following Fortran programs in the indicated order: `NTOOLE`, `CD1`, `T18`, and `T17`. These programs are described below.

Program name: `stripl` (strip version 1)

Purpose: Prepare a file, `ingam`, of deleted descriptions from a single MenuDriver database, with duplicates removed for reclassification.

Type: C shell script

Usage: `stripl intreeA`

Input: `intreeA` (as an example)

Output: `ingam`

Notes: As in `lib1s`, the text file `cntrl` must be present. `stripl` invokes the following Fortran programs in the indicated order: `NTOOLE` and `SHEAD`.

Program name: `strip2` (strip version 2)

Purpose: A subscript used by `lib1s` to invoke the Fortran programs.

Type: C shell script

Program name: NTOOLE (new tool E)

Purpose: Split a MenuDriver database into two parts, `headm` (the menu portion), and `int` (the descriptions, without MTABLE). Also prepare file `tbl1`, a table of lines where each line contains the name of a program, its library, and its GAMS classification. Each line of `tbl1` is associated with a description in `int`.

Type: Fortran program [source code: `ntoole.f` (located in `tem40.a`)]

Usage: `ntoole < in1`

Input: `in1` contains a single line, which must be the name of a MenuDriver database, such as `intreeA`.

Output: `tbl1`, `headm`, `int`, `lg1`

Notes: `tbl1` gives line numbers of where a description begins in file `int`. With the help of `cntrl` and this table, deletion of descriptions may occur. `headm` will become the front end of the new database after stripping. Ignore file `lg1` (the log file).

Program name: CNTRL (control file creator)

Purpose: To help prepare a control file, `cntrl`, used for stripping.

Type: Fortran program [source code: `cntrl.f` (located in `strip.a`)]

Usage: `a.out3`

Input: A MenuDriver database, such as `intreeA`. (The code prompts for the file name.)

Output: `cntrl`

Notes: `cntrl` is originally just a copy of `tbl1`. Using an editor such as `vi`, the user deletes from file `cntrl` the lines containing names *not* to be stripped. In fact, CNTRL is just a copy of NTOOLE blocking certain output files from being created.

Program name: CD1

Purpose: Strip descriptions from a MenuDriver database using control file `cntrl`.

Type: Fortran program [source code: `cd1.f` (located in `tem40.a`)]

Input: `int`, `tbl1`, and `cntrl`

Output: `int2`, and `int3`

Notes: `int2` contains the stripped descriptions, and `int3` contains what is left from `int`. Roughly the size of `int2` plus the size of `int3` equals the size of `int`.

Program name: SHEAD (strip descriptions and headers)

Purpose: Strip descriptions using `cntrl`, and remove duplicates and headers. The output file may be used for reclassification.

Type: Fortran program [source code: `shead.f` (located in `strip.a`)]

Input: `int`, `tbl1`, and `cntrl`

Output: `ingam` and `lg1`

Notes: This is a stripping code similar to CD1, except headers are removed as well as duplicate descriptions, and there is no analog of `int3`.

---

³ Because the output file is named `cntrl`, this cannot be the name of the CNTRL executable.

ingam differs from int2 in three respects: (1) It has an identification block, (2) GAMS headers are omitted, and (3) the description for a given routine will be given only once, regardless of how many times it appeared in cntrl. The lgl file will have useful information about duplicates, as well as copies of the descriptions that have *not* been stripped.

Program name: T18 (tool 18)

Purpose: Construct an MTABLE for int3 and place in front of int3 to make int4.

Type: Fortran program [source code: t18.f (located in tem40.a)]

Input: int3

Output: int4

Program name: T17 (tool 17)

Purpose: Insert headm (the menu portion of the original database) in front of int4 to make int5.

Type: Fortran program [source code: t17.f (located in tem40.a)]

Input: headm and int4

Output: int5

Notes: File int5 will become the new database, ntrees. The code updates the first line with the date of the MTABLE revision.

Most Fortran programs are kept in ar-file tem40.a, which is archived to the same directory as strip.a.

3.3.1.2. File Formats. For completeness, we give here detailed descriptions of the formats of some of the files that are used by the stripping tools. **Caution:** These tools examine only the first line of a database description, so take the first name of a set of equivalent names as the name of the routine. Since they currently use blank as the delimiter, any hyphenated type descriptor must be included in the name.⁴

File tbl1 contains one line for each routine in the description file, int, as follows:

- col. 1-21 NAME (CHARACTER*21), the name of the routine, as read from the first line of its database description.
- col. 22-27 LFIRST (INTEGER), the first line of its description in file int.
- col. 32-45 LIBNAM (CHARACTER*14), the name of the library containing the routine.⁵
- col. 46-63 GCODE (CHARACTER*18), its GAMS classification code.

The format of cntrl is assumed to be the same as tbl1, except that it is read with the equivalent of format (A,10X,2A) and the LFIRST field is ignored.⁶ No ordering is assumed here; deletion occurs in the order descriptions appear in int. Any of the three

⁴ It is thus safest to use the CNTRL code to produce a suitably formatted starting point for deletion specification.

⁵ The tools that write tbl1 are slightly incompatible with those which read cntrl. Columns 30-61 are actually written from a CHARACTER*32 variable that has the 16-character library name and GAMS code packed into it, with the first two characters blank. (Thus col. 62-63 will be blank.)

⁶ Actually, the last two fields are read into a single CHARACTER*32 variable and split apart by the code.

fields NAME, LIBNAM, GCODE may be blank. This "wild card" feature allows stripping all routines from a given library, for example, by including a line that has the library name in col. 32–45 and is otherwise blank. A routine can be completely removed from a given library by leaving col. 46–63 blank. **Caution:** A totally blank line strips everything!

File `ingam` consists of a sequence of MenuDriver descriptions, in the format described in Section 2.5.1. No ordering is assumed. The file created by SHEAD will be in the order of appearance in `int`, except that subsequent occurrences of an identical routine description will be omitted. The file output by SHEAD also has an initial identification block, consisting of the string " `ingam`" followed by a line of thirty hyphens. (See example below.) Additional information may be inserted before the line of hyphens, if desired.

3.3.1.3. A Example. The following is the `tbl1` file for the database `intreeA` given in Section 2.5.2.⁷ We note that it is consistent with the MTABLE counts that appear there.

F06BLE-S	3	NAG	A3
XADD-S	8	SLATEC	A3
XADJ-S	11	SLATEC	A3
XC210-S	14	SLATEC	A3
XCON-S	17	SLATEC	A3
XRED-S	20	SLATEC	A3
XSET-S	23	SLATEC	A3
BRENT-S	28	mss13	A3
EXRPACK-S	30	mss13	A3
A02ABE-S	35	NAG	A4
A02ACE-S	38	NAG	A4
C06GBE-S	41	NAG	A4
C06GCE-S	44	NAG	A4
C06GQE-S	47	NAG	A4
C06GSE-S	50	NAG	A4
F06CLE-S	53	NAG	A4
CARG-C	58	SLATEC	A4
R9PAK-S	62	SLATEC	A6
R9UPAK-S	65	SLATEC	A6
C06BAE-S	70	NAG	A7
C06GBE-S	73	NAG	A7
C06GCE-S	76	NAG	A7
C06GQE-S	79	NAG	A7
C06GSE-S	82	NAG	A7

As an example, we deleted lines from the above `tbl1` to produce the following `cntrl` file that requests deletion of three entries, one from NAG and two from SLATEC:

C06GCE-S	44	NAG	A4
R9PAK-S	62	SLATEC	A6
R9UPAK-S	65	SLATEC	A6

The resulting `int2` file from CD1 is:

C06GCE-S   Complex conjugate of complex sequence
C06GCF-D

SLATEC   GAMS: A6

⁷ This is the same as the `cntrl` file that would be output by the CNTRL code.

R9PAK-S   Pack a base 2 exponent into a floating point number.
D9PAK-D

R9UPAK-S   Unpack a floating point number X so that $X = Y*2^{**N}$.
D9UPAK-D

Note that, in addition to the requested three descriptions, this also contains one header, since we have deleted all of the SLATEC routines from category A6. This file contains exactly the lines noted as missing if we execute "diff int int3".

Following is the result of comparing the old and new databases. Note that category A6 has disappeared altogether, which will cause MenuDriver to malfunction if a user asks it to "show" routines in this category.

```
milstein% diff intreeA ntreeA
1c1
<      5      8      5      0      0      mdrive 10/07/93 FNF MTABLE  02/13/96
---
>      5      8      5      0      0      mdrive 10/07/93 FNF MTABLE  03/19/96
15c15
<      8      MTABLE:  02/13/96
---
>      7      MTABLE:  03/19/96
19,23c19,22
<      33      A4              7  NAG
<      56      A4              1  SLATEC
<      60      A6              2  SLATEC
<      68      A7              5  NAG
<      85              END
---
>      33      A4              6  NAG
>      53      A4              1  SLATEC
>      57      A7              5  NAG
>      74              END
67,69d65
< C06GCE-S      Complex conjugate of complex sequence
< C06GCF-D
<
82,89d77
<
<      SLATEC              GAMS:      A6
<
< R9PAK-S      Pack a base 2 exponent into a floating point number.
< D9PAK-D
<
< R9UPAK-S      Unpack a floating point number X so that  $X = Y*2^{**N}$ .
< D9UPAK-D
```

Note that, since we only asked to delete C06GCE from category A4, its description still appears in A7. (We could use the wild card feature to strip it out completely by blanking out the "A4" in the first line of cntrl.)

```
milstein% grep C06GCE intreeA
C06GCE-S      Complex conjugate of complex sequence
C06GCE-S      Complex conjugate of complex sequence
milstein% grep C06GCE ntreeA
C06GCE-S      Complex conjugate of complex sequence
```

The `ingam` file that is produced if we run SHEAD using the same `intreeA` and `cntrl` is as follows. Note that it includes an initial header block and has no GAMS header lines. (If we had deleted the same description from more than one category, it would appear only once here.)

```

ingam
-----
C06GCE-S      Complex conjugate of complex sequence
C06GCF-D

R9PAK-S       Pack a base 2 exponent into a floating point number.
D9PAK-D

R9UPAK-S      Unpack a floating point number X so that  $X = Y*2**N$ .
D9UPAK-D

```

3.3.2. Reclassification

To begin a reclassification process two files must be present: `ingam`, a file of descriptions to be classified or reclassified, and `gams`, a file of program names along with their GAMS classifications. An example will be given below. The purpose of this process is to prepare files of the given descriptions in a format suitable for insertion into MenuDriver databases by the tools described in Section 3.3.3.

Three scripts will be described: `lib1m` (library tool for MTABLE generation), `lib1r` (library tool for mass reclassification or classification), and `reclas3` (reclassification tool 3). The reclassification process can only be used for one library at a time, so that the names and GAMS classifications found in file `gams` must only belong to a single library.

These scripts and related files are stored in ar-file `reclass.a`. The version last updated on `yy-mm-dd` is saved to SCF archive directory `.m29/.nmg/unicos/mdriver/tools` as `reclass.yymmdd`.

3.3.2.1. Tool Descriptions. The first step of the process is to execute script `lib1m`. This script copies the current MenuDriver databases to the current directory. It then executes Fortran program `MTABLE`. This program strips the GAMS labels from the `MTABLE`'s of each database, and writes these labels into a file called `gamt` (GAMS total). The stripping process creates duplicate labels, which are removed at a later stage. The purpose of having the labels from the databases is for truncation of labels during the reclassification process. By this it is meant that the GAMS labels in file `gams` are full-length classification labels. They must be truncated to labels found in the present databases. Two possible errors might occur in the truncation process: the name of a description might not be found in file `gams`, and even if found it might be impossible to truncate the associated label to a label found in `gamt`.⁸ If these errors happen they are reported in an output diagnostics file.

⁸ This may be due to the fact that we are introducing a routine in a category for which we formerly had no routines available. (Empty terminal nodes are not allowed by the current MenuDriver.) It will be necessary to add this category via an update to the menu portion of the database, as described in Section 3.3.4, and then re-run `lib1m`.

The second step in this process is to execute script `lib1r`. This script creates insertion files and reports errors which are associated with the truncation process. `lib1r` will prepare insertion files for all databases where descriptions in `ingam` belong to those main GAMS categories. `lib1r` executes three Fortran programs for each database, `SHOWLB1`, `HDNG2`, and `HDNG4`. The execution of these programs is accomplished by script `reclas3`, which can be considered as a subscript of `lib1r`. Briefly, `SHOWLB1` creates a file `gamm` (GAMS MTABLE) which contains a subset of `gamt` for a specified database, with duplicates removed. `HDNG2` creates file `outh2`, a file of the descriptions found in `ingam`, where each description now has a header showing its library and its full GAMS label as found in `gams`. `HDNG4` creates file `int`, which is the insertion file; this file has truncated labels and is in alphabetic order.⁹ Thus `HDNG4` does truncation and sorting.

Program name: `lib1m` (library tool for MTABLE generation)

Purpose: Create file `gamt`, containing the GAMS labels found in the current MenuDriver databases.

Type: C shell script

Usage: `lib1m`

Input: Collection of MenuDriver databases in the directory in which `lib1m` is run. These must have names of the form `intree $\alpha$` , where α is a capital letter.

Output: `gamt`

Notes: The script executes Fortran program MTABLE.

Program name: `lib1r` (library tool for mass reclassification or classification)

Purpose: Prepare insert files for descriptions found in file `ingam`.

Type: C shell script

Usage: `lib1r lib`

(The script has one input argument, the library name.)

Input: `ingam`, `gams`, `gamt`

Output: `lg1` (the log file); `int2A`, `int2C`, ... (the insertion files); `outh2aA`, `outh2aC`, ... (output diagnostic files if errors have occurred)¹⁰

Notes: `lib1r` executes subscript `reclas3` which in turn executes Fortran programs.

Program name: `reclas3` (reclassification tool 3)

Purpose: Execute Fortran programs which prepare the insertion files. These programs are executed for each insertion file produced.

Type: C shell script

Usage: `reclas3 A lib`

(The script has two input arguments, the lead letter of the GAMS classification of a database and the library name.)

Input: `ingam`, `gams`, `gamt`

Output: `int2A`, `outh2aA` (see output of `lib1r`, above)

⁹ The truncated GAMS codes will be in lexicographic order and the routine names will be in alphabetical order within each GAMS classification.

¹⁰ There will be a series of output files for each main GAMS classification containing routines in `ingam`. The final letter will be the same as for the associated `intree`-file.

Notes: reclas3 executes Fortran programs SHOWLB1, HDNG2, and HDNG4 described below.

Program name: MTABLE (GAMS MTABLE extractor)

Purpose: Generate file gamt from MenuDriver databases. File gamt contains GAMS labels as found in the MTABLE's of the current databases.

Type: Fortran program [source code: mtable.f (located in main.a)]

Input: MenuDriver databases (as for lib1m, above)

Output: gamt contains GAMS labels as found in the MTABLE's of the current MenuDriver databases.

lg1, the log file, shows the dates for the floor databases.

Program name: SHOWLB1 (show labels for database)

Purpose: Produce file gamm, the subset of gamt for a particular database, with duplicates removed.

Type: Fortran program [source code: showlb1.f (located in pr10.a)]

Usage: showlb1 < in1

Input: gamt, in1

Output: gamm

Notes: in1 must contain two lines. Line one has the database's GAMS label (for intreeA, this would be "A"). The second line has the library name; for example, MSSSL. If SHOWLB1 is run stand-alone, then these lines are entered at the terminal; they are not prompted for.

Program name: HDNG2 (heading preparer 2)

Purpose: Prepare and associate headers for each description in ingam.

Type: Fortran program [source code: hdng2.f (located in pr10.a)]

Usage: hdng2 < in1

Input: in1, gams, ingam, gamm

Output: dgams, header, outh2, outh2a

Notes: in1 is as described above. dgams is a direct access file of descriptions. header is a file of headers associated with the descriptions in ingam. Each header contains the library name, the routine name, and the full GAMS label for the description as found in file gams. Multiple GAMS classifications will generate multiple headers for a routine. These are in one-to-one correspondence with the description blocks in outh2. outh2 is a file of descriptions, where each description now has its associated header (with full GAMS label). outh2a is a diagnostic file created if an error occurs.

Program name: HDNG4 (heading preparer 4)

Purpose: Sort the descriptions and truncate labels

Type: Fortran program [source code: hdng4.f (located in pr10.a)]

Input: gamm, header, and dgams (see above)

Output: int

Notes: `int` is the insertion file, with headers in the format required for GAMS header lines, as described in Section 2.5.1. (`reclas3` renames `int` according to the lead GAMS label, for example `int2A`, because this is the name expected when the user goes to the insertion process.)

The Fortran programs are kept in ar-files `main.a` and `pr10.a`, which are archived to the same directory as `reclass.a`.

3.3.2.2. Input File Formats. For completeness, we give here detailed descriptions of the formats of the input files that must be prepared for the reclassification tools.

The descriptions in file `ingam` are expected to be in the format described in Section 2.5.1. The file containing them must have a blank line after each description. No ordering is assumed. The `ingam` file described in Section 3.3.1.2, above, is in the required format. **Caution:** There *must* be a lead information block, terminated by a line that begins with 6 or more hyphens, as in the file generated by SHEAD.

The GAMS codes are read from a file `gams` that contains a list of routine names followed by one or more GAMS classification codes. The file *must* begin with an information block that is terminated by a line that begins with 6 or more hyphens. The following lines are assumed to be in the same basic format: a name followed by a slash-terminated list of associated GAMS classifications.¹¹ Some tools are more picky than others as to the exact columns occupied by these two fields. The file `gams.out` generated by EXDOC¹² uses format (1X,A8,7X,A):

col. 2–9 NAME, a routine name;

col. 17– CATLST, a list of one or more GAMS codes, each terminated by a slash.

As an example, the following is the GAMS file that was used when the FITGEN package was added to MSSL in September 1995:

```
GAMS categories for new FITGEN package.
Mon Sep 25 15:33:56 PDT 1995
File converted by GAMSCONV and codes made LC.
L8c1 -> L8c1b -- Fri Sep 29 11:42:43 (FNF)
```

```
-----
DFGDEMO      K1a1a3/K1a1b/L8c1b/
DFITGE       K1a1a3/L8c1b/
DLRENT       K1a1a3/L8c1b/
FITGEN        K1a1a3/L8c1b/
SFGDEMO      K1a1a3/K1a1b/L8c1b/
SFITGE       K1a1a3/L8c1b/
SLRENT       K1a1a3/L8c1b/
```

3.3.2.3. A Example. To illustrate reclassification, let us continue the example of section 3.3.1.3 by pretending that we wish to move the three routines deleted in that example to library MSSL. We will use the file `ingam` displayed there. The following `gams`

¹¹ **Caution:** The letters after the initial one in the GAMS code must be lower case.

¹² See Section 3.4.1.

file (partially shown) was obtained by randomly inserting fictitious lines for the three routines C06GCE, R9PAK, R9UPAK into the existing gams file for MSSSL.¹³

gams	MSSL	mdriver	GAMS	classifications	09/09/94	15:45:16	rp

BESEI0		C10b1/					
BESEI1		C10b1/					
BESEK0		C10b1/					
BESEK1		C10b1/					
R9PAK		A6b1/		(new line)			
BESI0		C10b1/					
BESI1		C10b1/					
BESJ0		C10a1/					
R9UPAK		A6b10/		(new line)			
BESJ1		C10a1/					
BESK0		C10b1/					
BESK1		C10b1/					
BESY		C10a3/					
BIMOND		E2a/					
BLU		D2a2/					
BSOL		D2a2/					
BTMS		D2a2/					
C06GCE		A4a11/A7b5/		(new line)			
CDRV		D2a4/					
DAW		C8/					
DEC		D2a1/					
DECBT		D2a2/					
DUMACH		R1/					
DVODE		I1a1/I1a2/					
...							

Here is the first part of the gamt file produced by lib1m. Next to it we display the complete file gamm produced by SHOWLB1. Note that not only does it contain only GAMS codes that begin with "A", but it has no duplicate labels. (This is the only gamm file, since all three of the routines in ingam are classified only in category A.)

.....	.....
gamt	gamm
.....	.....
A3	gamm intreeA
A3	-----
A3	A3
A3	A4
A4	A6
A4	A7
A4	
A6	
A6	
A7	
C	
C1	
C1	
C2	
C2	
C2	
C2	
...	

¹³ Although this illustrates the fact that ordering is not assumed in the gams input file, we keep the standard files sorted for maintenance purposes.

Following are the files header and outh2 produced by HDNG2 for this example. The information blocks were written by HDNG2. Note that there are two entries for C06GCE because it had two different GAMS codes in file gams.

header 03/28/96 17:49:43

```
-----
1  MSSL  C06GCE  GAMS:  A4a11
2  MSSL  C06GCE  GAMS:  A7b5
3  MSSL  R9PAK   GAMS:  A6b1
4  MSSL  R9UPAK  GAMS:  A6b10
```

outh2 03/28/96 17:49:43

```
-----
MSSL  C06GCE  GAMS:  A4a11

C06GCE-S  Complex conjugate of complex sequence
C06GCF-D

MSSL  C06GCE  GAMS:  A7b5

C06GCE-S  Complex conjugate of complex sequence
C06GCF-D

MSSL  R9PAK   GAMS:  A6b1

R9PAK-S  Pack a base 2 exponent into a floating point number.
D9PAK-D

MSSL  R9UPAK  GAMS:  A6b10

R9UPAK-S  Unpack a floating point number X so that  $X = Y \cdot 2^{**N}$ .
D9UPAK-D
```

Here is the final output file int (= int2A) produced by HDNG4. Note that labels from outh2 have been truncated to agree with gamm and the second block has been moved to its proper sorted position. These headers are now in the format required for a MenuDriver database, although the A6 header appears twice here.

```
MSSL  GAMS:  A4

C06GCE-S  Complex conjugate of complex sequence
C06GCF-D

MSSL  GAMS:  A6

R9PAK-S  Pack a base 2 exponent into a floating point number.
D9PAK-D

MSSL  GAMS:  A6

R9UPAK-S  Unpack a floating point number X so that  $X = Y \cdot 2^{**N}$ .
D9UPAK-D

MSSL  GAMS:  A7

C06GCE-S  Complex conjugate of complex sequence
C06GCF-D
```

3.3.3. Insertion

Starting with a set of insertion files, such as produced by a reclassification (Section 3.3.2), and the current set of MenuDriver databases, script `lib1i` (library tool for mass insertion) creates new databases with the provided descriptions inserted in the proper places.

These scripts and related files are stored in ar-file `insert1.a`. The version last updated on `yy-mm-dd` is saved to SCF archive directory `.m29/.nmg/unicos/mdriver/tools` as `insert1.yymmdd`.

3.3.3.1. Tool Descriptions. Script `lib1i` compiles the needed Fortran programs, executes subscript `insert2` once for each insertion file, and then removes unnecessary files. `insert2` executes five Fortran programs, which in turn prepare tables locating all descriptions in the old database and in the insertion file, do the insertion, build a new MTABLE, and concatenate the result onto the menu portion from the original database.

Program name: `lib1i` (library tool for mass insertion)

Purpose: Insert new descriptions into existing MenuDriver databases.

Type: C shell script

Usage: `lib1i`

Input: Collection of MenuDriver databases and associated insertion files in the directory in which `lib1i` is run. These must have names of the form `ntree $\alpha$` and `int2 $\alpha$` , respectively, where α is a capital letter.¹⁴ There must be a `ntree $\alpha$` file for each `int2 $\alpha$` , but the converse need not be true.

Output: Collection of updated MenuDriver databases, where `newtree $\alpha$` is the result of inserting from `int2 $\alpha$` into `ntree $\alpha$` .

Notes: `lib1i` executes subscript `insert2`, which in turn executes Fortran programs, for each `int2`-file.

Program name: `insert2` (insertion tool 2)

Purpose: Execute Fortran programs which actually do the insertion. These programs are executed for each new database produced.

Type: C shell script

Usage: `insert2 A`

(The argument is the lead letter of the GAMS classification of a database.)

Input: `int2A`, `ntreeA`

Output: `int5`, the new database. (`lib1i` renames this `newtreeA`.)

There are also other miscellaneous output files created by the Fortran programs, which are eventually destroyed by `lib1i`.

Notes: `insert2` executes Fortran programs NTOOLE, NTOOLF, CD2, T18 and T17, in that order. All but NTOOLF and CD2 were described in Section 3.3.1.1.

---

¹⁴ Note that the name `ntree $\alpha$` is consistent with the stripped databases produced by `lib1s` (Section 3.3.1.1), as is `int2 $\alpha$` with the insertion files produced by `lib1r` (Section 3.3.2.1).

Program name: NTOOLF (new tool F)

Purpose: Prepare file `tbl2`, a table of lines where each line contains the name of a program, its library, and its GAMS classification in `int2`.

Type: Fortran program [source code: `ntoolf.f` (located in `tem40.a`)]

Usage: `ntoolf < in2`

Input: `in2` contains a single line, which must be the name of an insertion file, such as `int2A`.

Output: `headm2`, `int6`, `tbl2`

Notes: Ignore file `headm2`. `int6` is not used; it is a copy of the file named in `in2`. (NTOOLF is basically a copy of NTOOLE blocking certain output files from being created and renaming others.)

Program name: CD2

Purpose: Merge `int1` and `int2` using `tbl1` and `tbl2`.

Type: Fortran program [source code: `name.f` (located in `tem40.a`)] or C shell script

Input: `int1`, `int2`, `tbl1`, `tbl2`

`int1` is the file `int` produced by NTOOLE; `int2` is the insertion file.

`tbl1` and `tbl2` are the tables produced by NTOOLE and NTOOLF, respectively.

Output: `int3`, the merged description portion of the new database.

Notes: If `int2` contains duplicate headers, all but the first will be eliminated. If a header in `int2` already appears in `int1`, the corresponding description will be merged into `int3` in proper alphabetical order.

The Fortran programs are kept in ar-file `tem40.a`, which is archived to the same directory as `insert1.a`.

3.3.3.2. Input File Formats. For completeness, we give here a detailed descriptions of the formats of the insertion files, in case they must be prepared by hand, rather than using the reclassification tools.

`int2` contains a series of MenuDriver descriptions. Each description must be terminated by a blank line and preceded by a header line followed by another blank line. The header contains the library and GAMS classification code and must be in the format described in Section 2.5.1. At present, the GAMS codes must be truncated as they appear in the current MenuDriver databases. These description blocks must be in lexicographic order by GAMS code. Within a given classification, descriptions must be alphabetical by routine name.¹⁵ See Section 3.3.2.3 for a sample of the required format.

Although insertion is usually done one library at a time, the tools allow multiple libraries. If this is the case, the description blocks within a given classification must be alphabetical by library.

---

¹⁵ This is a convention, to make it easier for users to find their way through a set of descriptions, rather than a hard-and-fast rule.

3.3.3.3. A Example. To illustrate insertion, let us continue the example of Section 3.3.2.3 by inserting the contents of file `int2A` prepared there into the database `ntreeA` produced as the result of the deletion described in Section 3.3.1.3. The `tbl1` produced by NTOOLE will be similar to the one listed in Section 3.3.1.3, but updated for the three deletions. The `tbl2` file produced by NTOOLF is:

C06GCE-S	3	MSSL	A4
R9PAK-S	8	MSSL	A6
R9UPAK-S	13	MSSL	A6
C06GCE-S	18	MSSL	A7

Following is the result of comparing the two databases:

```

milstein% diff ntreeA newtreeA
1c1
<      5      8      5      0      0      mdrive 10/07/93 FNF MTABLE  03/19/96
----
>      5      8      5      0      0      mdrive 10/07/93 FNF MTABLE  04/02/96
15c15
<      7      MTABLE:  03/19/96
----
>      10     MTABLE:  04/02/96
19,22c19,25
<      33     A4              6  NAG
<      53     A4              1  SLATEC
<      57     A7              5  NAG
<      74             END
----
>      33     A4              1  MSSL
>      38     A4              6  NAG
>      58     A4              1  SLATEC
>      62     A6              2  MSSL
>      70     A7              1  MSSL
>      75     A7              5  NAG
>      92             END
54a58,62
>             MSSL              GAMS:  A4
>
> C06GCE-S      Complex conjugate of complex sequence
> C06GCF-D
>
77a86,98
>
>             MSSL              GAMS:  A6
>
> R9PAK-S      Pack a base 2 exponent into a floating point number.
> D9PAK-D
>
> R9UPAK-S      Unpack a floating point number X so that X = Y*2**N.
> D9UPAK-D
>
>             MSSL              GAMS:  A7
>
> C06GCE-S      Complex conjugate of complex sequence
> C06GCF-D

```

To further illustrate what we have done in this series of examples, following are the MTABLE's for the original and final versions of the database. Note that three new MSSL blocks have appeared and one SLATEC block has been removed.

```

::::::::::::::::::
intreeA
::::::::::::::::::
      8      MTABLE:  02/13/96
      1      A3          1  NAG
      6      A3          6  SLATEC
     26      A3          2  mssl3
     33      A4          7  NAG
     56      A4          1  SLATEC
     60      A6          2  SLATEC
     68      A7          5  NAG
     85              END

::::::::::::::::::
newtreeA
::::::::::::::::::
     10      MTABLE:  04/02/96
      1      A3          1  NAG
      6      A3          6  SLATEC
     26      A3          2  mssl3
     33      A4          1  MSSL
     38      A4          6  NAG
     58      A4          1  SLATEC
     62      A6          2  MSSL
     70      A7          1  MSSL
     75      A7          5  NAG
     92              END

```

3.3.4. Menu Fixing

In the event that a new library or package includes routines in a GAMS classification that is not included in the current MenuDriver databases, it will also be necessary to fix the menu portion of the affected databases. This is accomplished by using the tools described in Section 3.2.

For completeness, these scripts and related files are also stored in ar-file `menuf.a`, and the version last updated on `yy-mm-dd` is saved to SCF archive directory `.m29/.nmg/unicos/mdriver/tools` as `menuf.yymmdd`.

3.3.5. Comparison

There are two scripts, `cmp1` and `cmp1a`, that may be used to compare two MenuDriver databases. Both of these are stored in `cmpare.a`. The version last updated on `yy-mm-dd` is saved to SCF archive directory `.m29/.nmg/unicos/mdriver/tools` as `cmpare.yymmdd`.

Program name: `cmp1` (compare)

Purpose: Compare the description portions of two MenuDriver databases with the same lead GAMS classification letter.

Type: C shell script

Usage: `cmp1 oldtree newtree`

Input: `oldtree` and `newtree` are the names of the two databases to be compared (`intreeA` and `newtreeA`, for example).

Output: `lg1` (log file)

Notes: This script uses Fortran programs `NTOOLE`, `NTOOLF`, and `CMR2`.¹⁶ It reports the names of descriptions found in one database but not the other, and it shows discrepancies in descriptions which describe the same routine. Since the librarian archives older versions of a database, this script is useful to find out the differences between an older database and a new one, possibly one which has just been created (`newtreeA` stands for "new `intreeA`").

¹⁶ The first two have been described earlier. `CMR2` appears below.

Program name: `cmp1a` (alternate compare)

Purpose: Compare the description portions of two MenuDriver databases with the same lead GAMS classification letter.

Type: C shell script

Usage: `cmp1a oldtree newtree`

Input: Same as above.

Output: `lg1` (log file)

Notes: This script uses Fortran programs NTOOLE1 and NTOOLF1. (These are slightly modified versions of NTOOLE and NTOOLF, with different formats for the generated lists. They will not be described further here.) `cmp1a` reports the names of descriptions found in one database but not the other. This script invokes `diff`. Consequently, the output looks different than that from `cmp1`.

Program name: `CMPR2` (description compare)

Purpose: Perform the actual comparison for `cmp1`.

Type: Fortran program [source code: `cmpr2.f` (located in `tem40.a`)]

Input: `headm`, `headm2`, `tbl11`, `tbl2`

Output: `lg1` (log file)

Notes: The input files are created by NTOOLE and NTOOLF.

3.3.6. Other Tools

Following are descriptions of some miscellaneous tools that are useful for MenuDriver database maintenance.

The tool T17A is useful in checking for possible problems after an insertion or deletion has been performed, or after a menu fix has been done.

Program name: T17A

Purpose: Check for consistency of menu nodes with MTABLE.

Type: Fortran program [source code: `t17a.f` (located in `menuf.a`)]

Input: All 14 MenuDriver data trees, `intreeA`, ..., `intreeZ`, are assumed to be in the same directory as the T17A executable.

The code prompts for the user's initials to label the output.

Output: `lg1` (A diagnostic file)

Notes: The script will fail if all trees are not present. To check just a few, copy all of the "public" `intree`-files to the current directory first and then overwrite the new ones to be checked.

The output file will contain the first lines of the input files, for identification purposes, followed by diagnostic information about any problems found.

Following is the `lg1` file produced when T17A was run using the `intreeA` file created in Section 3.3.1.3 instead of the true `intreeA`:

```
*
*      Program t17a.f:                      04/15/96    15:53:34    rpd
*
*
```

```

intreeA      mdrive 10/07/93 FNF MTABLE 03/19/96 ← (new one)
intreeC      mdrive 10/07/93 FNF MTABLE 12/06/95
intreeD      mdrive 08/09/95 FNF MTABLE 12/06/95
intreeE      mdrive 10/07/93 FNF MTABLE 12/20/95
intreeF      mdrive 10/07/93 FNF MTABLE 12/06/95
intreeG      mdrive 08/10/95 FNF MTABLE 12/06/95
intreeH      mdrive 10/07/93 FNF MTABLE 12/14/95
intreeI      mdrive 10/07/93 FNF MTABLE 12/06/95
intreeJ      mdrive 10/07/93 FNF MTABLE 12/06/95
intreeK      mdrive 10/07/93 FNF MTABLE 12/14/95
intreeL      mdrive 08/10/95 FNF MTABLE 12/06/95
intreeN      mdrive 04/29/94 FNF MTABLE 12/06/95
intreeR      mdrive 10/07/93 rpd MTABLE 12/07/95
intreeZ      mdrive 10/07/93 rpd MTABLE 12/14/95

```

A6 not in MTABLE; error.

Note that T17A caught the empty category A6 that was created with that deletion step. (A6 is a terminal node in the menu portion of the database.) This tool will also detect cases where categories mentioned in MTABLE do not appear in the menu portion.

3.4. Extracting Information from SLATEC-Format Source Codes

This section describes tools that are available for extracting information from SLATEC-format source codes. Note that SLATEC, MATHLIB and PMATH are entirely in SLATEC format, and many of the newer MSSL routines are as well.

3.4.1. Use of EXDOC to Extract On-Line Documentation

The Fortran program EXDOC is useful for extracting the prologues from SLATEC-format source codes. EXDOC expects that the input file (`slasrc`) will be in SLATEC/LDOC standard form, as described in [1].¹ The current version also accepts `/*DECK` as an alternative to `*DECK`, to allow for C codes with SLATEC-format prologues as comment blocks. It extracts the prologues and writes them as is, with column 1 blank, to the output (`sladoc`) file. The log file (`exdoc.log`) will contain information about all `*DECK` lines read, as well as warning and error messages. The latter contain one or more `'>'` characters.²

EXDOC assumes that only documentation for user-callable routines is wanted. If a `C***SUBSIDIARY` line is present, indicating an internal routine, the prologue will be omitted. The output includes the lines between `*DECK` and `C***BEGIN PROLOGUE` (should be the `SUBROUTINE` or `FUNCTION` statement), as well as the `*DECK` line itself and an optional header, which is read from the terminal. The `*DECK` lines can be used by a postprocessor to separate write-ups, if desired. (The output will contain `*DECK` even though the input may have had `/*DECK`.)

Although the SLATEC standard limits routine names to six characters, EXDOC picks up eight from the `C***BEGIN PROLOGUE` line to accommodate certain MATHLIB and MSSL routines. *Caution:* No check is made as to whether the name on the `*DECK` line agrees with that on the `C***BEGIN PROLOGUE` line.

Side Effects: EXDOC will create a scratch file, `exdoc.temp`, which will be destroyed on normal termination.

EXDOC prompts the user for the names of the input and output files. Perhaps the simplest way to describe its use is with an example. Here we exercise EXDOC on its own source code. (As usual, responses by the user are in **boldface**.)

```
milstein% exdoc
```

```
SLATEC Document Extraction Program -- Version 07
```

```
Type name of source file (or return for "slasrc").  
exdoc07.f
```

```
Type name of output file (or return for "sladoc").  
exdoc.doc
```

```
Type name of log file (or return for "exdoc.log").
```

¹ EXDOC is a simplified version of the SLTOLD program described in [1]. It only checks for `*DECK`, `C***BEGIN PROLOGUE`, `C***SUBSIDIARY` and `C***END PROLOGUE` lines, and it does no reformatting of the text.

² File names in parentheses in this paragraph are the defaults that are used if the user does not supply them.

exdoc.doc.log

```

Reading source file exdoc07.f
Writing documentation to file exdoc.doc
Log file is exdoc.doc.log

```

Type header line (or return for no header).

```

EXDOC
LASTNB
SLPKUP
SLREAD
SLWRIT

```

```

753 lines read from input file (including 5 *DECK lines).
95 lines written to output file (containing 1 routine documents).

```

>>> Check log file for 1 error message(s).

A maximum of 40 characters will be read for the source and output file names, 32 for the log file name. If the user wishes a header line of some kind to be inserted between the *DECK line and the document proper, it should be typed after the "Type header line" prompt. (Here we simply typed a carriage return, indicating that no header is wanted.)

Note that EXDOC prints a routine name at the terminal each time a *DECK line is read. At the end is a summary indicating the number of lines read from the source file, the number of *DECK lines encountered, and the number of routine documents written to the output file. Note that only one routine was user-callable. The terminal output also indicates that an error message has been generated. To see it, we display the log file:

```
milstein% cat exdoc.doc.log
```

```
SLATEC Document Extraction Program -- Version 07
```

```

Reading source file exdoc07.f
Writing documentation to file exdoc.doc
Log file is exdoc.doc.log

```

Header is:

```

*DECK EXDOC read at source file line 2
Processing completed for EXDOC document.

```

```
*DECK LASTNB read at source file line 502
```

```
*DECK SLPKUP read at source file line 518
```

```

>>> Error: *DECK line read while processing LASTNB prologue.
Reinitializing.

```

```
SLPKUP is subsidiary. Document omitted.
```

```

*DECK SLREAD read at source file line 615
SLREAD is subsidiary. Document omitted.

```

```

*DECK SLWRIT read at source file line 683
SLWRIT is subsidiary. Document omitted.

```

```
Received i/o status indicator = -1 when reading file exdoc07.f.
```

Terminating normally.

```
753 lines read from input file (including 5 *DECK lines).
95 lines written to output file (containing 1 routine documents).
1 errors detected. (See >>> lines above.)
```

We see that, as with the terminal output, a message is written each time a *DECK line is read. This is followed by an indication of the disposition of the document for that routine, except in the case of LASTNB. It turns out that this routine is not in SLATEC format (it has no C***BEGIN PROLOGUE line, even though it does begin with a *DECK line).

To illustrate the processing that is done on the source code to produce the output document, we look at the first few lines of both the input and output files:

```
milstein% head -15 exdoc07.f
C2 4 6 8(1)2 4 6 8(2)2 4 6 8(3)2 4 6 8(4)2 4 6 8(5)2 4 6 8(6)2 4 6 8(7)2
*DECK EXDOC
C***BEGIN PROLOGUE EXDOC
C***PURPOSE SLATEC document extraction routine -- Version 07.
C          EXDOC is a code to extract documentation from a source code
C          in SLATEC/LDOC form.
C***LIBRARY (NONE)
C***CATEGORY S1
C***TYPE CHARACTER
C***KEYWORDS DOCUMENTATION, CONVERSION, UTILITY
C***AUTHOR Fritsch, Fred N., (LLNL)
C          Livermore Computing, Mathematical Software Service
C          (510)422-4275, fnf@llnl.gov
C***DESCRIPTION
C
milstein% head -15 exdoc.doc
*DECK EXDOC
***BEGIN PROLOGUE EXDOC
***PURPOSE SLATEC document extraction routine -- Version 07.
          EXDOC is a code to extract documentation from a source code
          in SLATEC/LDOC form.
***LIBRARY (NONE)
***CATEGORY S1
***TYPE CHARACTER
***KEYWORDS DOCUMENTATION, CONVERSION, UTILITY
***AUTHOR Fritsch, Fred N., (LLNL)
          Livermore Computing, Mathematical Software Service
          (510)422-4275, fnf@llnl.gov
***DESCRIPTION

*Usage:
milstein% tail -5 exdoc.doc
940505 Increased length of FIN to allow for long chain names. (FNF)
960313 Also increased length of FOUT and FLOG, to facilitate use
      from a script. Also updated comments to be consistent with
      actual declarations. (FNF)
***END PROLOGUE EXDOC
```

We see that the document for EXDOC begins with the *DECK line and ends with the C***END PROLOGUE line.³ We also see that the comment character has been replaced

---

³ Note: Since some REVISION HISTORY sections are quite long, you may wish to remove or shorten this section for the final document.

with a blank for better readability. Had a nonempty header line been typed, it would appear verbatim immediately following the *DECK line.

Had there been more than one user-callable routine in the source file, all of their documents would be merged in the output file. Each document begins with a *DECK line, so it is possible to separate them into individual files with an editor or suitable script that searches for the *DECK pattern. We illustrate this by processing a file containing the complete source code for the FITGEN package:

```
milstein% exdoc
```

```
SLATEC Document Extraction Program -- Version 07
```

```
Type name of source file (or return for "slasrc").
```

```
fitgen.f
```

```
Type name of output file (or return for "sladoc").
```

```
fitgen.doc
```

```
Type name of log file (or return for "exdoc.log").
```

```
fitgen.doclog
```

```
Reading source file fitgen.f
```

```
Writing documentation to file fitgen.doc
```

```
Log file is fitgen.doclog
```

```
Type header line (or return for no header).
```

```
--> Sample documentation header line <--
```

```
FITGEN
```

```
DFGDEMO
```

```
DFGPRT
```

```
DFNDEM
```

```
DGDEMO
```

```
DFITGE
```

```
DLRENT
```

```
SFGDEMO
```

```
SFGPRT
```

```
SFNDEM
```

```
SGDEMO
```

```
SFITGE
```

```
SLRENT
```

```
2233 lines read from input file (including 13 *DECK lines).
```

```
1075 lines written to output file (containing 7 routine documents).
```

```
milstein% grep DECK fitgen.doc
```

```
*DECK FITGEN
```

```
*DECK DFGDEMO
```

```
*DECK DFITGE
```

```
*DECK DLRENT
```

```
*DECK SFGDEMO
```

```
*DECK SFITGE
```

```
*DECK SLRENT
```

```
milstein% head fitgen.doc
```

```
*DECK FITGEN
```

```
--> Sample documentation header line <--
```

```
SUBROUTINE FITGEN
```

```
***BEGIN PROLOGUE FITGEN
```

```
***PURPOSE Fit a general linear form to given data.
```

```
***LIBRARY MSSL
```

```
***CATEGORY K1A1A3, L8C1
```

```

***TYPE      PACKAGE (FITGEN-P)
***KEYWORDS  GENERAL LINEAR FITTING, LEAST SQUARES
***AUTHOR    Fritsch, Fred N., (LLNL/LC/MSS)

```

In this case, the source file contains 13 routines, of which seven are user-callable. We also see where the header line appears in the output file. (There is a copy of it after each *DECK line in `fitgen.doc`.)

3.4.2. The `runexdoc` Script

Since we will generally want the source code separated into different files, one for each subprogram, for insertion into the appropriate source archive, it is generally more efficient to generate the documents after the source code has been split via UNIX utility `fsplit`. A script `runexdoc` has been written to simplify this process. It takes two directories as arguments. It runs `EXDOC` on all `.f`-files in the source directory (`srcdir`) and puts the output files in the documentation directory (`docdir`), without the `.f` extension. It also creates a file `srcdir.log` containing the merged `exdoc.log` files. *Caution:* Any existing file by the name `exdoc.in`, `exdoc.log`, `exdoc.names`, or `srcdir.log` will be destroyed.

We illustrate this whole process by re-generating the documents for FITGEN:

```

milstein% mkdir fitgensrc
milstein% cd fitgensrc
milstein% fsplit ../fitgen.f
fitgen.f
dfgdemo.f
dfgppt.f
dfndem.f
dgdemo.f
dfitge.f
dlrent.f
sfgdemo.f
sfgprt.f
sfndem.f
sgdemo.f
sfitge.f
slrent.f
milstein% ls
./          dfgppt.f      dgdemo.f      sfgdemo.f     sfndem.f
../         dfitge.f      dlrent.f      sfgprt.f      sgdemo.f
dfgdemo.f   dfndem.f      fitgen.f      sfitge.f      slrent.f
milstein% cd ..
milstein% runexdoc fitgensrc fitgendoc
Creating directory fitgendoc

Type header line for documents.
(( FITGEN Documentation -- Version 1.1 ))

13 files processed
7 documents created
Look at fitgensrc.log for details
milstein% ls fitgendoc
./          dfgdemo  dlrent    sfgdemo    slrent
../         dfitge   fitgen    sfitge

```

Note that these seven files contain the same documents (except for different header lines) that were written to file `fitgen.doc` in the previous example. We illustrate their contents by looking at the first few lines of three of them.

```
milstein% head -5 fitgendoc/*fit*
==> fitgendoc/dfitge <==
*DECK DFITGE
(( FITGEN Documentation -- Version 1.1 ))
    SUBROUTINE DFITGE (FNFORM, RPAR, IPAR, XDATA, NDIM, YDATA, NDATA,
+                     NTERMS, WEIGHT, COEFF, RSD2, WORK, JOB, IERR)
***BEGIN PROLOGUE  DFITGE

==> fitgendoc/fitgen <==
*DECK FITGEN
(( FITGEN Documentation -- Version 1.1 ))
    SUBROUTINE FITGEN
***BEGIN PROLOGUE  FITGEN
***PURPOSE  Fit a general linear form to given data.

==> fitgendoc/sfitge <==
*DECK SFITGE
(( FITGEN Documentation -- Version 1.1 ))
    SUBROUTINE SFITGE (FNFORM, RPAR, IPAR, XDATA, NDIM, YDATA, NDATA,
+                     NTERMS, WEIGHT, COEFF, RSD2, WORK, JOB, IERR)
***BEGIN PROLOGUE  SFITGE
```

Caution: As a general rule, we do not include `*DECK` lines in documents we make available in NMG documentation archives. Since we do not yet have a tool that strips these out, it is currently necessary to edit them out by hand. A better alternative would be to include an option in `EXDOC` to not write the `*DECK` line to the output file.⁴

3.4.3. Use of EXINFO to Extract Purposes and GAMS Codes

The Fortran program `EXINFO` is useful for extracting purposes and GAMS codes from SLATEC-format source codes in a form suitable for input to the MenuDriver reclassification tools (see Section 3.3.2). This tool, like `EXDOC`, assumes the input is in standard SLATEC/LDOC format [1]⁵ and only generates output for user-callable routines.

`EXINFO` is rather simpler than `EXDOC` in that it reads from `stdin` and writes the purposes to `stdout`, which facilitates use from a script. It is generally used as:

```
exinfo < info.in > info.out
```

The output GAMS codes and log file have fixed names (`gams.out` and `exinfo.log`, respectively). `EXINFO` will abort if a file by either of these names exists prior to execution.

We use the same `FITGEN` source file as before as an example:

```
milstein% exinfo < fitgen.f > fitgen.purp
milstein% tail -5 exinfo.log
Terminating normally.
```

⁴ We probably do not want to make this the *only* option, to allow for cases where we still want to run `EXDOC` on files containing multiple SLATEC-format prologues, such as the `PMATH` C-coded modules.

⁵ Actually, it only checks `*DECK`, `C***SUBSIDIARY`, `C***PURPOSE`, `C***TYPE`, `C***CATEGORY` and `C***END PROLOGUE` lines.


```

2233 lines read from input file (including 13 *DECK lines).
24 lines written to output file (containing 7 routine descriptions).
7 lines written to file gams.out

```

The end of the log file tells us what to expect in the output. Here are the files:

```

milstein% cat gams.out
FITGEN      Klala3/L8c1/
DFGDEMO     Klala3/Klala3/L8c1/
DFITGE      Klala3/L8c1/
DLRENT      Klala3/L8c1/
SFGDEMO     Klala3/Klala3/L8c1/
SFITGE      Klala3/L8c1/
SLRENT      Klala3/L8c1/

milstein% cat fitgen.purp
FITGEN-P     Fit a general linear form to given data.

SFGDEMO-S    FITGEN bivariate demonstration program.
DFGDEMO-D

SFITGE-S     Fit a general linear form to given data.
DFITGE-D     Finds the linear combination of user-supplied basis
              functions that is the best least-squares fit to a given
              set of data points.

SLRENT-S     "FNFORM" routine for Laurent series fitting by FITGEN.
DLRENT-D

SFGDEMO-S    FITGEN bivariate demonstration program.
DFGDEMO-D

SFITGE-S     Fit a general linear form to given data.
DFITGE-D     Finds the linear combination of user-supplied basis
              functions that is the best least-squares fit to a given
              set of data points.

SLRENT-S     "FNFORM" routine for Laurent series fitting by FITGEN.
DLRENT-D

```

File `fitgen.purp` illustrates a problem with the current procedure. In the case of equivalent routines, such as `SFITGE` and `DFITGE`, there will be duplicate entries in the output purpose file, one from each source routine. In this example, there are only four distinct purposes. To avoid problems using the MenuDriver reclassification tool, it would be best to edit the file to remove duplicates. (Easy in this case: just delete the last three purposes.)

3.4.4. The `runexinfo` Script

If we have already separated the source code into different files, as above, we would use script `runexinfo` to run `EXINFO` on all `.f`-files in the source directory (`srcdir`). The combined descriptions will appear in file `srcdir.desc`, and the GAMS classification codes will appear in file `srcdir.gams`. The script also creates a file `srcdir.ilog` containing the merged `exinfo.log` files.⁶ *Caution:* Any existing file by the above names or `exinfo.log`, `exinfo.names`, or `gams.out` will be destroyed.

⁶ Note that this log file deliberately has a different name from the one generated by `runexdoc`.

We illustrate this process by re-generating the purposes and GAMS codes for FITGEN:

```
milstein% mv exinfo.log fitgen.infolog
milstein% mv gams.out fitgen.gams
milstein% runexinfo fitgensrc
Extracted descriptions will appear in file fitgensrc.desc
GAMS classifications will appear in file fitgensrc.gams
```

```
13 files processed
7 GAMS records created
Look at fitgensrc.ilog for details
```

```
milstein% cat fitgensrc.gams
GAMS classifications extracted from .f files in directory fitgensrc
runexinfo done on -- Thu Mar 14 16:05:47 PST 1996
```

```
-----
DFGDEMO      K1a1a3/K1a1b/L8c1/
DFITGE       K1a1a3/L8c1/
DLRENT       K1a1a3/L8c1/
FITGEN       K1a1a3/L8c1/
SFGDEMO      K1a1a3/K1a1b/L8c1/
SFITGE       K1a1a3/L8c1/
SLRENT       K1a1a3/L8c1/
```

We notice two differences from the original `gams.out` (now in `fitgen.gams`). First, `runexinfo` has written a header that documents where these GAMS codes came from and when. Second, the names are now alphabetical, because that is the way the script picks them up, rather than the order of the source codes in `fitgen.f`.

There is a similar difference in the output purpose files, but both have the same duplicate entry problem:

```
milstein% diff fitgen.purp fitgensrc.desc
1,2c1,3
< FITGEN-P      Fit a general linear form to given data.
<
---
> Descriptions extracted from .f files in directory fitgensrc
> runexinfo done on -- Thu Mar 14 16:05:47 PST 1996
> -----
12a14,15
>
> FITGEN-P      Fit a general linear form to given data.
```

References

- [1] Fritsch, F. N., *SLATEC/LDOC Prologue: Template and Conversion Program*, UCID-21357, Rev.1 (November 1988).

3.5. Superfile Generation

This section describes tools and procedures that may be used for generating the superfiles that are needed to support source code fetching. The reader is assumed to be familiar with the contents of Section 2.6.4.

3.5.1. Hand-generated Superfiles

For inserting the source code for a small number of routines into a library, it may be possible to generate the associated superfiles by hand. Just remember that a superfile must contain a complete call list, not just the routines called directly by the routine being inserted. In cases where the new routine calls routines already in the library, it may be necessary to concatenate the superfiles for called routines (and remove duplicate entries if necessary).

As an example, when the FITGEN package was added to MSSSL the superfiles were generated from the "ROUTINES CALLED" lists in the source codes. For those routines that call XERRWV or XERRWD, the superfiles were completed by adding copies of the appropriate superfile. Since some of these routines call LINPACK routines, which are in SLATEC, the associated entries were deleted from the superfiles and comments were added to the source files to warn users to load with the SLATEC library.¹

3.5.2. Generating Superfiles for SLATEC-format Routines

A script `ssg` (SLATEC superfile generator) has been written to extract preliminary superfile information from the "ROUTINES CALLED" lists in SLATEC-format source codes.² There is also a driver script, `bssg`, which executes `ssg` on all files in a directory and produces files with names of the form `routine.final`. These are not proper superfiles because (1) the name of the routine itself is not present, and (2) they do not include routines called by lower-level routines in the call tree.

A simple script `sgfix` is provided to correct problem (1), and to remove "(none)" from the superfiles for terminal nodes in the call tree. For simple cases, it may be sufficient to concatenate existing superfiles, as described above, to solve problem (2). The scripts described here may be found in `ocfkms` directory `~pang_nmg/script`, `milstein` directory `/usr/local/math/nmg/tools`, or SCF archive directory `.m29/.nmg/unicos/tools`.

For a complete solution to problem (2), one should copy the `routine.final` files produced by `ssg`, with the ".final" suffix removed, to another directory, say `dir1`, copy the superfiles for called routines³ there, and then run the `supergen` code:

---

¹ `fetch` cannot handle cross-library calls, and we do not want to duplicate code by adding these routines to MSSSL.

² **Caution:** There must be a single routine per file for `ssg` to work properly, and the files should *not* have `.f` extensions.

³ The easiest way to do this is to extract the entire contents of the superfile archive for the library into which the routine(s) are to be inserted.

```
supergen dir1 dir2
```

where `dir2` is the directory into which the updated superfiles are to be written. (*Caution:* `dir2` must not exist prior to executing `supergen`.) `supergen` will give error messages to indicate routines for which superfiles have not been provided. This approach is valid for cases, such as the SLATEC library itself, where the "ROUTINES CALLED" lists have been independently been determined to be correct.

`supergen` may be found in `ocfkms` directory `~pang_nmg/bin`. The source code for this C-coded program may be found in `ocfkms` directory `~pang_nmg/develop/supergen`, or SCF archive directory `.m29/.nmg/unicos/tools`.⁴

3.5.3. Using `sfg`

A much more comprehensive superfile generator, called `sfg` (superfile generator), has been written to use a C-coded Fortran parser to generate complete superfiles based on examination of the source code itself. The C shell script can handle either complete source files or `ar`- or `tar`-format archives that contain the individual source files. It requires `parse` and `supergen` (described above).

`sfg` and `parse` are in `ocfkms` directory `~pang_nmg/bin`. The source code for `parse` may be found in `ocfkms` directory `~pang_nmg/develop/parse`. Copies of `sfg` and the `parse` source code have been stored in SCF archive directory `.m29/.nmg/unicos/tools`, but may be of limited utility, since `sfg` has built-in references to files in `ocfkms` directory `~pang_nmg` (actually referred to as `~` there, assuming that it is being run by its author, Dan Pang).

---

⁴ *Caution:* This requires include files that appear to be specific to the IBM AIX system.

3.6. How to Install a Library in NMG

The purpose of this section is to review the steps needed to completely install a new library in NMG. Most of this information is also relevant to installing a new version of an existing library. It is assumed that the reader is familiar with the contents of Chapter 2.

3.6.0. Installation Overview

The complete installation of a library in NMG involves the following steps:

1. Prepare on-line documentation
2. Prepare source code
3. Prepare superfiles
4. Prepare legal notices
5. Update MenuDriver databases
6. Update Advisor databases (if appropriate)
7. Validate
8. Check helpfiles for necessary updates
9. Notify users

Note: Steps 2–4 are required only if source code is being made available (as for MSSL, PMATH and SLATEC). If source code is to be fetchable, it is usually easier to create the source archive first and then use existing tools to extract information necessary for the documentation archive and MenuDriver databases.

Caution: The `nmg.tar` file (see Section 2.8) should also be updated whenever such a change has been made.

In the following, `$nmgdir` is the value of a C shell script variable that contains the directory chain indicating where NMG is installed. On most NMG platforms, this has been set via:

```
set nmgdir = /usr/local/math/nmg
```

Caution: If an entirely new library is being added, this will require an update to the `fetch` script, `$nmgdir/fetch/fetch4`, to include the new library and the associated files. (Read the initial script comments for details.) A complete list of the file names currently in use is included in the Appendix, to avoid duplication when adding a new library.

3.6.1. Documentation

On-line documentation is read from an ar-format file containing individual routine documents. The name of the file is the same as the routine name (lower case). Typically, the contents¹ of a documentation archive will be a strict subset of the associated source archive (if available). These ar-files are in `$nmgdir/fetch/doc`.

---

¹ That is, the list of names of files.

For document fetching, obtain on-line documentation for each user-callable routine in the library from the Fortran source code or elsewhere.² (We do *not* supply on-line documentation for subsidiary routines.) At a minimum, these documents should describe each argument in the calling sequence for each routine. It may also be useful to provide overview documentation for packages of related routines contained within a library.

For SLATEC-format source codes, the utility EXDOC can be run to extract documents directly from the prologues, and the script `runexdoc` is helpful in automating part of the process. (See Section 3.4 for more information.) *Note:* If source code is also being made available, it is usually more efficient to prepare the source archive first (see below).

3.6.2. Source Code

On-line source code is read from an `ar`-format file containing source codes for individual routines, one routine per file.³ The name of the file is the same as the routine name (lower case).⁴ These `ar`-files are in `$nmgdir/fetch/src`.

The UNIX utility `fsplit` may be useful in splitting a large Fortran code into its individual routines. Our `unefem` script can then be used to remove the `.f`'s from the file names.

3.6.3. Superfiles

"Superfiles" are text files containing the complete call tree for each routine for which source code is available. The first line of a superfile contains the name of the routine itself.⁵ This is followed by a list of subsidiary routines, one name per line. Terminal nodes in the call tree will have only a single line in their superfiles. (See Section 2.6.4 for more information.)

The superfiles are maintained in an `ar`-format file containing one routine per file. The name of the file is the same as the routine name (lower case). There is a one-to-one correspondence between files in a source archive and those in the associated superfile archive.⁶ These `ar`-files are in `$nmgdir/fetch/super`.

The tools described in Section 3.5 may be used to generate these for Fortran routines. It is sometimes necessary to do hand-fixes to the superfiles, for example to include co-routines (routines not called by the requested routine, but which the user must call to do initialization or other related work). Any addition to or deletion from a source archive requires a superfile archive update.

---

² In the case of a minor revision, these documentation files will require little or no change. For a major revision, it is generally easier to generate a whole new set of documents.

³ This one-routine-per-file rule is violated in MSSSL, where sometimes whole collections of closely-related routines are contained in one file.

⁴ See Section 2.6.4 for information on C-coded modules.

⁵ Superfiles for user-callable entries defined in C-coded modules will have the name of the associated `.c`-file as the first entry.

⁶ Exception: User-callable entries defined in C-coded modules will appear only in the superfile archive, but not in the associated source archive.

3.6.4. Legal Notices

In order to convey ownership and distribution information, a legal notice is appended to the beginning of each source file delivered by the "fetch" script. These files are in `$nmgdir/fetch/legal`.

If a new version of an existing library is installed, a minor legal notice update may be required. If a new library is added, a legal notice must be prepared. (Pattern it after the most appropriate existing notice.)

3.6.5. MenuDriver Databases

The MenuDriver provides access to a categorized list of routine purposes, which are split into subtrees (also referred to as databases) by the top level of the GAMS classification scheme. These files are in `$nmgdir/mdriver`. This directory also contains the file `menu`, which is the text for the MenuDriver main menu. The first line of this file contains an alphabetized list of the libraries which are available on the machine on which these databases are installed. This line will need to be updated if a new library is being added. (See Section 2.5 for more information.)

Locally-written tools (see Section 3.3) are used to update the MenuDriver databases to include information for a new library. In order to add, delete, or replace routines, a list of routine names with associated library and GAMS classification(s) is required. For additions or replacements, new purposes are also needed. For SLATEC-format source codes, the utility `EXINFO` can be run to extract purposes and GAMS codes directly from the prologues, and the script `runexinfo` is helpful in automating part of the process. (See Section 3.4 for more information.)

3.6.6. Advisor Databases

The Advisor provides access to a decision tree for advice on selection of numerical software. The decision tree is split into subtrees (also referred to as databases) by the top level of an arbitrary classification scheme. These files are in `$nmgdir/advisor`. (See Section 2.4 for more information.)

Locally-written tools (see Section 3.2) are used to update the Advisor databases. (These tools may also be used to update or add information nodes to the classification scheme of a MenuDriver database; for example, we may need to include categories for which we had no available routines before this new library was installed.⁷) Advisor database updates may be needed when routines are added or deleted. If this is a new version of an existing library, check whether any recommended routines have been superseded, and change the advice accordingly. As time permits, it would also be a good idea to determine whether the new library provides abilities not previously included that should be mentioned in the Advisor databases or whether an enhanced routine should now be recommended over a routine from another library. (This will likely require nontrivial testing by a knowledgeable person.)

---

⁷ Such an update must be done *before* adding the purposes for these routines to the MTABLE part of the database.

3.6.7. Library Validation

A collection of tools has been written to validate the consistency of the various databases and archive files that support NMG. The appropriate validation procedure (see Chapter 4) should be run after any nontrivial update to a library or database. Any discrepancies should be corrected before putting the revised files "on the floor". If a new library is added, the validation procedure will need to be updated to include it.

3.6.8. Helpfile Updates

The installation of a new library or new version of an existing library should be reflected in the "Library Availability" and "Library Abbreviations" help screens. Depending on the nature of the changes, other help screens may also need to be updated as a result of a library update. (See Section 3.1 for helpfile update procedures.)

3.6.9. User Notification

The final step in any installation project is notifying the users of NMG that something has been changed. At the minimum, an entry should be added to the beginning of the nmgnws file, which starts with the date the updated file(s) were installed (in yy-mm-dd format) and contains a brief description of the purpose of the change. In case of significant changes, a system news notice should be generated, as well.

3.6.Appendix. Library-*fetch*-file Correspondence Table

The following table shows in one place all libraries that are currently supported by *fetch* and the associated file names. (This is provided to help avoid duplication when naming files for a new library.)

Library name	Documentation archive name	Source code archive name	Superfile archive name	Legal notice
MATHLIB	mathdoc	---	---	---
MSSL (class 1-2)	mssldoc	mssl	superm	legalm
MSSL (class 3)	---	mssl3	super3	legal3
NAG	nagdoc	---	---	---
PMATH	pmathdoc	pmath	superp	legalp
SLATEC	sladoc	slatec	supers	legals

3.7. How to Add or Delete a Routine in a Library

The purpose of this section is to review the steps needed to add a routine to or delete one from an existing library in NMG. It is assumed that the reader is already familiar with the material in Section 3.6.

Caution: The `nmg.tar` file (see Section 2.8) should also be updated whenever such a change has been made.

3.7.1. Adding a Routine

Adding a routine to an existing library in NMG involves the following steps:

1. Prepare on-line documentation
2. Prepare source code
3. Prepare superfiles
4. Update MenuDriver databases
5. Update Advisor databases (if appropriate)
6. Validate
7. Notify users

Note: Steps 2–3 are required only if source code is being made available (as for MSSSL, PMATH and SLATEC). If source code is to be fetchable, it is usually easier to prepare the source code first and then use existing tools to extract information necessary for the documentation archive and MenuDriver databases.

1. Documentation. Obtain on-line documents for each user-callable routine being added to the library and add them to the appropriate documentation archive. At a minimum, these documents should describe each argument in the calling sequence for each routine. For SLATEC-format source codes, the utility EXDOC can be run to extract documents directly from the prologues, and the script `runexdoc` is helpful in automating part of the process. (See Section 3.4 for more information.) *Note:* If source code is also being made available, it is usually more efficient to prepare the source archive first (see below).

2. Source Code. The first step in preparing source code for insertion into the appropriate source archive is to make sure it is organized one routine per file.¹ The UNIX utility `fsplit` may be useful in splitting a large Fortran code into its individual routines. Our `unefem` script can then be used to remove the `.f`'s from the file names.

3. Superfiles. It is also necessary to prepare a "superfile" corresponding to each of the files being added to the source archive and add these to the associated superfile archive. In the case of simple file or package insertion, these can be prepared by hand. Otherwise, the tools described in Section 3.5 may be used to generate these. It is sometimes necessary to do hand-fixes to the superfiles, for example to include co-routines (routines not called by

---

¹ In MSSSL and `mssl3` we sometimes distribute an entire package in one file.

the requested routine, but which the user must call to do initialization or other related work).

4. MenuDriver Databases. In order to add new user-callable routines, it is necessary to add them to the appropriate list of routine names and GAMS classification(s). New purposes are also needed. For SLATEC-format source codes, the utility EXINFO can be run to extract purposes and GAMS codes directly from the prologues, and the script `runexinfo` is helpful in automating part of the process. (See Section 3.4 for more information.) Then use the appropriate tools (see Section 3.3) to add these new routines to the MenuDriver databases.

5. Advisor Databases. Check whether the addition of these new routines requires modification to Advisor recommendations, or possibly addition of a new branch to a decision tree. If so, use the appropriate tools (see Section 3.2) to update the Advisor databases.

6. Library Validation. The appropriate validation procedure (see Chapter 4) should be run after any nontrivial update to a library or database. Any discrepancies should be corrected before putting the revised files "on the floor".

7. User Notification. The final step in any installation project is notifying the users of NMG that something has been changed. At the minimum, an entry should be added to the beginning of the `nmgnews` file, which starts with the date the updated file(s) were installed (in `yy-mm-dd` form) and contains a brief description of the purpose of the change. In case of significant changes, a system news notice should be generated, as well.

3.7.2. Deleting a Routine

Deleting a routine to an existing library in NMG involves the following steps:

1. Remove its documentation from the appropriate documentation archive.
2. If source code has been fetchable, remove its source code and superfile from the appropriate archives. Be sure to also delete any subsidiary routines that are not used by other, non-deleted routines. (Failure to do so will be signaled by a report of "orphans" by the validation procedure.)
3. Remove its entries from any MenuDriver databases that mention it. (For completion, also remove it from the associated GAMS code file.)
4. Update Advisor databases (if appropriate) to avoid recommending a non-existent routine.
5. Validate.
6. Notify users. (A simple `nmgnews` entry should be sufficient.)

3.7.3. Replacing a Routine

Replacing a routine in an existing library in NMG involves the following steps:

1. Update its documentation in the appropriate documentation archive (if necessary).

2. If source code has been fetchable, replace its source code in the appropriate source archive. Check whether any associated subsidiary routines also require updating. Also check whether the associated superfile(s) need to be changed.
3. Update its MenuDriver databases entries as necessary
4. Validate.
5. Notify users. (A simple nmgnews entry is usually sufficient.)

3.8. Processing NMG Usage Statistics

This section describes the tools and procedures used to process NMG usage statistics.

3.8.1. How Statistics are Generated

As discussed in the descriptions of the NMG Main Control and Fetch Scripts in Chapter 2, a NMG statistics record is generated every time NMG is executed. This consists of a simple e-mail message that contains the login name of the user and the machine on which NMG was run. It indicates which commands were executed and which routines were requested via **fetch**. In the latter case, error messages are also written to the statistics record. The detailed statistics produced by **fetch** are generated in case we find a problem with a routine and need to contact users, or to collect statistics about usage of a particular library.

As an example, following are the three statistics records generated by the terminal session that produced the examples in Chapter 1.

```
Date: Mon, 18 Mar 1996 15:30:56 -0800
From: fritsch@milstein.anduin.ocf.llnl.gov (Frederick N. Fritsch)
To: fnf@llnl.gov
Subject: NMG statistics
```

```
DATE    Mon Mar 18 15:29:53 PST 1996 (milstein)
NEWS
MAIL
END
```

```
----- Mail message from utility NMG -----
USER: milstein:fritsch    DATE: Mon Mar 18 15:29:53 PST 1996
-----
```

This is a sample NMG mail message

```
Date: Mon, 18 Mar 1996 15:32:14 -0800
From: fritsch@milstein.anduin.ocf.llnl.gov (Frederick N. Fritsch)
To: fnf@llnl.gov
Subject: NMG statistics
```

```
DATE    Mon Mar 18 15:31:05 PST 1996 (milstein)
HELP
END
```

```
Date: Mon, 18 Mar 1996 15:36:43 -0800
From: fritsch@milstein.anduin.ocf.llnl.gov (Frederick N. Fritsch)
To: fnf@llnl.gov
Subject: NMG statistics
```

```
DATE    Mon Mar 18 15:32:23 PST 1996 (milstein)
ADVICE
MENU
FETCH
```

```
Viewed dpchim document from slatec
Saved dpchim document from slatec
Fetched dpchim source from slatec
Saved dpchfe document from slatec
```

```

    Fetched dpchfe source from slated
    Error: no dranf source in slated
  FETCH
    Retrieved pmath_rnf.c for dranf source from pmath
    Retrieved pm_params.h for dranf source from pmath
    Retrieved pm_rnfset.h for dranf source from pmath
    Retrieved pmath_cnv.c for dranf source from pmath
    Retrieved pm_cnvset.h for dranf source from pmath
    Fetched dranf source from pmath
  FETCH
    Fetched xermsg source from mssl
  END

```

Note that "Error" messages are generally the result of user errors, most frequently requesting a non-existent routine. These are not generally cause for concern, although repeated "Error" messages in a single session or several closely spaced in time may indicate that a user is having trouble using `fetch` and would initiate a personal contact (usually via e-mail). On the other hand, a "Trouble" message means that some file that `fetch` requires is missing or has become corrupted, requiring immediate action.

On the Cray Y-MP, these statistics records are sent to a special user `m29`, which results in the message being appended to file `/usr/local/math/nmg/nmgmail`. This special user has not been set up on the Cray J90's, so the messages are directed to user `fritsch`, who must manually save them to the indicated file. The command `m29mail`, aliased to `"mailx -f /usr/local/math/nmg/nmgmail"`, is run periodically to read recent statistics records for possible problems.

On `milstein`¹ and `ocfkms`, the statistics records are mailed to `fnf@llnl.gov`, who examines them as they arrive and accumulates them by hand for later processing.

3.8.2. Monthly Statistics

On the Cray's, on or shortly after the first of each month, the statistics for the previous month are saved to a file in `/usr/local/math/nmg/stats` named `sllymmmm`, where:

- `l` is a letter indicating the machine on which the statistics were generated (`y` for Cray Y-machine, `j` for Cray J90 cluster, etc.);
- `yy` is the last two digits of the year;
- `mmm` is a three-letter month designator (`jan`, `feb`, etc.).

When the previous month's statistics have been saved, the corresponding messages are deleted from `/usr/local/math/nmg/nmgmail`.

These files are all accumulated on one Cray (currently `luke`) and the `nmgpact` utility is run to compact the records. For example, on February 2, 1996, the command

```
nmgpact s*96jan > janstats
```

was run. The contents of `janstats` provides the overall usage counts reported in the MSS Quarterly Report:

---

¹ The open Sun SPARCstation that MSS uses as a development platform.

```

-----
sj96jan
-----
1 total message(s) processed
1 compacted record(s) written to sj96jan.pact
-----
sy96jan
-----
2 total message(s) processed
2 compacted record(s) written to sy96jan.pact

```

To illustrate the disk space saved by nmgpact, the following is the result of the command "wc s*96jan*":

24	89	603	sj96jan
14	59	364	sj96jan.pact
49	170	1126	sy96jan
30	106	680	sy96jan.pact
117	424	2773	total

The .pact-files only are saved each month in SCF archive directory .m29/.nmgstat/unicos/monthly.

3.8.3. Quarterly Statistics

At the same time the monthly statistics for the last month of a quarter are accumulated, a quarterly summary is also generated. The three monthly statistics files are concatenated to a file named slyyqn, where *l* and *yy* are as above and *n* is 1, 2, 3, or 4.

This, too, is compacted before being saved in SCF archive directory .m29/.nmgstat/unicos/quarterly. As an example, the following commands were run on January 10, 1996:

```

cat su95oct > su95q42
cat sy95oct sy95nov sy95dec > sy95q4
nmgpact s*95q4 > q4stats
countem > counts95q4

```

Note that, since the months do not conveniently occur in alphabetical order, one must explicitly list the monthly record files in the order in which they should appear in the quarterly accumulation.³

The last command invokes a simple C shell script that counts the number of entries in all .pact-files in the current working directory, so it is necessary to first delete any records from previous quarters in order to produce a correct result. This provides a check of the monthly ???stats file results and gives all in one place (but not in a very convenient format) the numbers for the MSS Quarterly Report. These counts-files are also saved in .m29/.nmgstat/unicos/quarterly. Here is counts95q4:

```

su95oct: 5
su95q4: 5

```

² The U-machine was turned off the end of October 1995.

³ Note: sy95q4.pact, for example, should be the same as "cat sy95oct.pact sy95nov.pact sy95dec.pact".

```
sy95dec: 9
sy95nov: 6
sy95oct: 7
sy95q4: 22
```

Statistics processing on the open side is even more manual. Statistics records from milstein and ocfkms are saved as they arrive in anduin file ~fritsch/mail/stat-milstein or -ocf, respectively. At the beginning of each quarter the messages from the previous quarter are saved to smyyqn or soyyqn, respectively. These are then sent to milstein, where they are stored in directory /usr/local/math/nmg/stats. (Note that there is no archival back-up for this directory.) When the previous quarter's statistics have been saved, the corresponding messages are deleted from the ~fritsch/mail/stat-files.

To simplify summary counting, "mail -f file > file.h" is executed to generate the output from mail's "h" command. This must be followed by "q" to terminate mail. (If there are more than 30 messages, we would need to do "h 31" before "q" to get the full list.) This is then edited slightly and examined by hand to determine the numbers for these platforms. (This could be automated, but the numbers have been sufficiently small that it hasn't seemed worthwhile.) As an example, here is so95q4.h:

```
mailx version 5.0 Fri Mar 18 15:04:31 PST 1994 Type ? for help.
```

```
"so95q4": 28 messages
```

```
>O 1 iotero@ocfkms.llnl Tue Oct 10 16:20 21/781 NMG statistics
O 2 maiden@ocfkms.llnl Wed Oct 11 13:41 23/846 NMG statistics
O 3 maiden@ocfkms.llnl Wed Oct 11 19:46 27/937 NMG statistics
O 4 maiden@ocfkms.llnl Wed Oct 11 19:53 22/813 NMG statistics
O 5 chase@ocfkms.llnl Fri Oct 13 10:56 23/851 NMG statistics
O 6 free@ocfkms.llnl.g Fri Oct 13 11:08 33/995 NMG statistics
O 7 free@ocfkms.llnl.g Mon Oct 16 13:37 23/840 NMG statistics
O 8 ychoi@ocfkms.llnl Mon Oct 16 14:31 23/847 NMG statistics
O 9 ychoi@ocfkms.llnl Mon Oct 16 14:33 22/808 NMG statistics
O 10 free@ocfkms.llnl.g Wed Oct 18 09:50 20/764 NMG statistics
O 11 greene@ocfkms.llnl Fri Oct 20 17:43 23/818 NMG statistics
O 12 ychoi@ocfkms.llnl Fri Nov 3 19:48 26/914 NMG statistics
O 13 free@ocfkms.llnl.g Wed Nov 8 16:17 30/967 NMG statistics
O 14 chase@ocfkms.llnl Thu Nov 9 11:47 23/841 NMG statistics
O 15 chase@ocfkms.llnl Wed Nov 22 23:40 24/857 NMG statistics
O 16 ychoi@ocfkms.llnl Thu Nov 30 22:57 21/772 NMG statistics
O 17 ychoi@ocfkms.llnl Thu Nov 30 22:58 23/843 NMG statistics
O 18 hascott@ocfkms.lln Sat Dec 2 13:28 24/856 NMG statistics
O 19 Charles J. Cerjan Fri Dec 8 07:17 21/801 NMG statistics
O 20 Russell G Berger Wed Dec 13 15:47 26/924 NMG statistics
O 21 Russell G Berger Wed Dec 13 15:55 21/807 NMG statistics
O 22 Russell G Berger Wed Dec 13 15:59 22/812 NMG statistics
O 23 Russell G Berger Wed Dec 13 16:11 21/807 NMG statistics
O 24 Russell G Berger Wed Dec 13 16:25 36/1145 NMG statistics
O 25 James W. Greene Thu Dec 14 17:24 23/840 NMG statistics
O 26 Charles J. Cerjan Fri Dec 22 14:56 23/850 NMG statistics
O 27 Yumi Choi Thu Dec 28 05:55 23/797 NMG statistics
O 28 Yumi Choi Thu Dec 28 05:58 29/1158 NMG statistics
```

```
?
```

Using grep, or by visual inspection, we see that the ocfkms numbers for the fourth quarter of 1995 are: Oct=11, Nov=6, Dec=11.

3.8.4. Tool Abstracts

Following are brief descriptions of tools used in conjunction with NMG statistics records.

Program name: countem (count them)

Purpose: Count the number of usage statistics records in all .pact-files in the current working directory.

Type: C shell script [located in /usr/local/math/nmg/stats on the Cray's]

Usage: countem > countfile

Input: All files in current directory with extension .pact.

Output: countfile, the number of statistics records in each file.

Notes: The script actually counts the number of occurrences of the pattern "From: ".

Program name: nmgpact (NMG statistics compactor)

Purpose: Eliminate extraneous lines from NMG usage statistics mail messages.

Type: perl script [located in /usr/local/math/nmg/stats on the Cray's]

Usage: nmgpact filelist > logfile

Input: filelist, a list of names of files containing NMG usage statistics messages.

Output: file.pact is the compacted version of file.

If file contains any mail messages, these are written to file.mail.

logfile contains a summary of the number of messages found in each file.

Notes: This script requires files Constants to define some global constants.

Program name: nmg.arch (NMG statistics archiver)

Purpose: Write copies of /usr/local/math/nmg/nmgmail to the SCF archive.

Type: perl script [located in /usr/local/etc (Cray Y-MP only)]

Usage: insert

Input: /usr/local/math/nmg/nmgmail

Output: SCF archive file .m29/.nmgstat/unicos/save/mail.1.yyymmdd, a copy of the nmgmail file from machine 1 as it existed on yy-mm-dd.

Notes: This chron job, which is executed once a week, has not yet been put in place on the Cray J90's. It was created to guard against loss of all NMG usage statistics in the event of disk failure on the Cray. (This has never happened!)

3.9. Back-up Procedures

This section describes the back-up directories and procedures used by NMG maintainers.

3.9.1. File Naming Conventions

As a general rule, back-up copies of NMG files are stored with suffices of the form *yymmdd* to indicate the version that was created/updated on day *dd* of month *mm* in year *yy*. In the case of files that normally have a suffix (such as *.a* or *.f*) the suffix is generally dropped. In the case of scripts, there may or may not be a "." before the date. For example, the most recent version of *adriver.f* is called *adriver.960227*, because it was updated February 27, 1996, and the executable created from it is called *adrv.960227*. The most recent version of *comparem* is *comparem960304*. In the case of *ar-* or *tar-*files, a long-list file ("*ar tv*" or "*tar tvf*") is often stored, so that the contents of the archive can be determined without having to transfer a large file from archival storage. (There may be an additional letter added to indicate special platform versions, such as *y* for Cray Y-MP, *j* for Cray J-90, *o* for the OCF IBM cluster, etc.)

Exceptions: (1) Historically, because its name is so short, the *.f* is not dropped from *help.f*, so we find *help.f.960109*; (2) back-ups for *nmg.tar* are stored as *nmgtar.yymmdd* (with list-file *nmglst.yymmdd*); (3) the *newnmg* and *fetch4* scripts are stored by version number rather than date, such as *fetch4.14h*.

3.9.2. Cray Directory Structure

The NMG files are maintained in directory */usr/local/math/nmg* on both the Cray Y-MP and the J90's.¹ This is a shared directory with read/write access for members of group *mss* (*m29* on the Y-MP). The structure is basically the same as described in Chapter 2, except that each directory generally contains a subdirectory *work* that contains back-up files. In addition, */usr/local/math/nmg* has the following subdirectories:

- /nmglib* — contains the current working version of NMGLIB;
- /tardir* — used to build up the current *nmg.tar*;
- /tarsave* — contains several back-up copies of *nmg.tar*.

3.9.3. SCF Archive Directory Structure

Most of the NMG back-up files are contained in shared directory *.m29/.nmg* on the SCF archive. Its structure has many parallels with the Cray structure and is as follows. Most directories contain README files describing their contents.

---

¹ There is a parallel structure on *milstein*, the open Sun SPARCstation we use as a development platform. On the OCF IBM cluster, NMG resides in *~pang_nmg/math/nmg*, and */usr/local/bin/nmg* is a soft link to *~pang_nmg/math/nmg/nmg*.

/documents — nmg.l and nmg.man back-ups.

/tarfiles — nmg.tar back-ups.² See file Notes for more information.

/unicos — top level for the main back-up structure.³ It contains the following:

- /advisor/databases — Advisor database back-ups;
- /advisor/program — Advisor source and executable back-ups;
- /advisor/tools — Advisor maintenance tools.
- /ctrl — NMG main control back-ups:
  - newnmg.yymmdd — the Main Control Script;
  - nmg.yymmdd — the nmg executable (for /usr/local/bin);
  - nmgdr.yymmdd — the source code for same: nmgdr.c;
  - nmgnews.yymmdd — nmgnews back-ups (incomplete set).
- /fetch/doc — documentation archive back-ups;
- /fetch/legal — legal notice back-ups;
- /fetch/src — source archive back-ups;
- /fetch/super — superfile archive back-ups;
- /fetch/work — Fetch Script back-ups.
- /help/screens — Helpfile back-ups;
- /help/program — Help Processor source and executable back-ups;
- /help/tools — Helpfile maintenance tools.
- /mdriver/databases — MenuDriver database back-ups;
- /mdriver/program — MenuDriver source and executable back-ups;
- /mdriver/tools — MenuDriver maintenance tools.
- /nmglib — NMGLIB back-ups:⁴
  - nmgsrsrc.yymmdd — ar-file containing contents of subdirectory /nmgsrsrc;
  - nmglst.yymmdd — the result of "ar tv nmgsrsrc.yymmdd";
  - nmgsrsubs.yymmdd — created via "tar cvf nmgsrsubs.tar nmgsrsubs/*";
  - lstsubs.yymmdd — the result of "tar tv nmgsrsubs.yymmdd".
- /tools — miscellaneous tools.

Details of the contents of many of the subdirectories are given in the following subsections.

Note: There is a much more fragmentary NMG back-up on the "open" side. The shared FAST archive directory .mss/.nmg contains only the following subdirectories:

- /ctrl — newnmg and fetch4 back-ups.
- /ocfkms — a structure similar to /unicos above, to back up the version installed on the OCF IBM cluster.
- /tarfiles — nmg.tar back-ups.

² Many of the files in nmg.tar are duplicated elsewhere in this back-up structure.

³ There used to also be a subdirectory /nltss for the old NMG.

⁴ These are (working) copies of files from nmg.tar directory /nmglib (see Section 2.8).

3.9.4. Advisor and MenuDriver Databases

Directories `/unicos/advisor/databases` and `/unicos/mdriver/databases` have similar contents. `dbasea.yymdd` contains the Advisor databases as they existed on `mm/dd/yy`. This is an ar-format file containing `menua`, the current `intree`-files, and `README`. The latter should have a line added at the end each time `dbasea.a` is updated, giving a brief explanation as to which databases were updated and why. File `dbasem.yymdd` contains similar information for MenuDriver databases. Depending on the context, `dblst.yymdd` is the "ar tv" listing for the associated database archive.⁵

3.9.5. Helpfiles

Directory `/unicos/help/screens` contains back-ups for the NMG helpfiles. `hlpscrn.yymdd` is the ar-format file described in Section 3.1. `lstscrn.yymdd` is the result of "ar tv `hlpscrn.yymdd`". Some of the tools in `/unicos/help/tools` are also contained in these `hlpscrn`-files.

3.9.6. Fetch Back-up Files

Most of the contents of the subdirectories of `/unicos/fetch` is self-explanatory. The only thing that may not be evident is the naming convention used for the "ar tv" listings.

In subdirectory `/doc`:

<code>liblst.yymdd</code>	is the listing for	<code>libdoc.yymdd</code>

In `/super`:

<code>lsupn.yymdd</code>	is the listing for	<code>supern.yymdd</code>

The correspondence in `/src` is less obvious:

<code>lstmss.yymdd</code>	is the listing for	<code>mssl.yymdd</code>
<code>lstms3.yymdd</code>	is the listing for	<code>mssl3.yymdd</code>
<code>lstpmath.yymdd</code> ⁶	is the listing for	<code>pmath.yymdd</code>
<code>lstsla.yymdd</code>	is the listing for	<code>slatec.yymdd</code>

3.9.7. GAMS Files

Files containing the GAMS classification codes required by the MenuDriver reclassification tool are maintained in shared SCF archive directory `.m29/.gams`. Its subdirectories are:

`/external` — GAMS classifications for various libraries, as obtained from Ron Boisvert (NIST) or other external source;

`/mss` — GAMS classifications for various libraries, created and/or updated by MSS personnel;⁷

`/scheme` — text defining the complete GAMS Classification Scheme. (Read file `.m29/.gams/README` for more details.)

⁵ Sorry, the same file name is used for both `dbasea.a` and `dbasem.a` listings, in different directories.

⁶ To be consistent, this should probably have been called `lstpma.yymdd`.

⁷ Use a file from `/external` only if directory `/mss` contains no file for the desired library.

Most files in `/external` and `/mss` have names of the form `libm.gamsn`, where:

- `lib` is the name of the library;
- `m` (if present) is an integer indicating the library version;
- `n` (if present) is an integer to distinguish various updates to the classifications.

For example, `nag16.gams2` is the second version of the GAMS classifications for Mark 16 of the NAG library.

Note that the internal formats of these files are not uniform, so look at a file before using it. In many cases there will be a header record, terminated by a line of 20 or more hyphens, indicating the history of the file.

3.9.8. Where Are the Tools?

Back-up storage for NMG-related tools is, unfortunately, rather disorganized, and the same tool may sometimes be found in different directories. Because of this, it is not always obvious where to find the latest version of a particular tool. There is a tool glossary at the end of this chapter that may help locate a particular tool.

Following is a (probably non-exhaustive) list of directories to look in on the SCF archive:

- `.m29/.nmg/tarfiles` (subdirectory `nmglib/tools` of `nmg.tar`)
- `.m29/.nmg/unicos/advisor/tools`
- `.m29/.nmg/unicos/help/tools`
- `.m29/.nmg/unicos/mdriver/tools`
- `.m29/.nmg/unicos/tools`
- `.m29/.utility`

Examine the `README` file in each directory and/or archive file for more details.

3.9.9. Usage Statistics

NMG usage statistics are saved in shared SCF archive directory `.m29/.nmgstat/unicos`. Its subdirectories are:

- `/monthly` — monthly usage reports;
- `/quarterly` — quarterly usage reports;
- `/yearly` — yearly usage summaries (rarely generated);
- `/tools` — back-ups for usage statistics processing tools;
- `/save` — temporary storage for back-up copies of Y-MP file `/usr/local/math/nmg/nmgmail`, which is where files mailed to user `m29` wind up on these machines.

Refer to Section 3.8 for more information.

3.10. Notes on Future Development

This final section contains notes on desirable future development for the NMG maintenance tools. Refer to Section 2.9 for related information.

3.10.1. What Kinds of Updates Need to Be Performed?

- Remove *list* from *lib*. [This involves document, source, and superfile archives. In order to make sure that we also get subsidiary routines, so as to not create orphans, we need to access all of the superfiles and remove anything that is not called by routines **not** being removed. (We don't want to just remove everything in the union of the superfiles, because there may be some subsidiaries, like XERROR, that are called by many unrelated routines.) A "remove" also involves MenuDriver databases. If any removed routines are recommended by Advisor, a red flag needs to be raised!]
- Insert *list* into *lib*. [Must supply necessary files for document, source, and superfile archives as well as descriptions and GAMS codes for MenuDriver databases.]
- Move *list* from *lib1* to *lib2*. [This is a combination of a "remove" and "insert", in which things removed are saved for the insert. Advisor databases may also need library name change.]
- Replace MenuDriver descriptions for *list*. [The library needs to be specified somehow. If GAMS headers aren't supplied, need to ask for library name. In this case, each description should be replaced wherever it appears in the MenuDriver databases. (This is an issue only for multiply-classified routines.) If GAMS headers are supplied, should this be interpreted as a reclassification, which should delete all occurrences first? (If so, should produce a list of categories description was in previously, as a check.)]

3.10.2. What Other Tools Would Be Helpful?

- In order to maintain archive integrity, some procedure should be devised for automatic version date updating for help screen files, so that we do not have to rely on this being done manually. (See Section 3.1.)
- We need a facility which would make it easier to include up-to-date platform-dependent information in the Library Availability help screen (and possibly others).
- We should provide features for subtree extraction and merging, to enable more massive decision tree revision than is convenient with the current Advisor tools. We should also make the NELIM option interactive. (See Section 3.2.)
- We should develop more portable and more reliable tools for generating superfiles. (See Section 3.5.)
- Given a list of names and a superfile archive, generate the union of the superfiles of the listed names, sorted and with duplicates removed. (Error message for missing names.)

- Given a list of names and a superfile archive, produce the list of references to nonexistent routines that would be generated if these routines were deleted. (Use with the output from the previous tool to determine which names should be left out of the deletion list.)
- If we go to a master database, we will need to develop tools to extract the information needed to construct the Advisor or MenuDriver databases and the **fetch** archives from the master database.

3. Appendix. Tool Directory

Because there are so many tools described in this chapter, we have created the following table to assist the database maintainer in locating a particular tool. The *Reference* column contains a reference to the section in which more information about this tool may be found. If the tool is stored in an ar- or tar-format archive, its name appears in the *Archive* column.

<i>Tool Name</i>	<i>Type</i>	<i>Purpose</i>	<i>Reference</i>	<i>Archive</i>
artotar	C shell script	Convert ar-file to tar-file	2.8.8	nmg.tar
assmb1	C shell script	Combine listed files into a single file	2.8.8	nmg.tar
bssg	C shell script	Driver for ssg	3.5.2	
catem	C shell script	Concatenate files with headers	3.1.2	
CD1	Fortran	Strip descriptions from a MenuDriver database for lib1s	3.3.1	tem40.a
CD2	Fortran	Merge new routines into existing descriptions for lib1a	3.3.3	tem40.a
CHANGE	Fortran	Maintain multiple versions in one source file	2.7, 2.8.8	nmg.tar
cmp1	C shell script	Compare description portions of two MenuDriver databases	3.3.4	compare.a
cmp2	C shell script	Compare description portions of two MenuDriver databases	3.3.4	compare.a
CMR2	Fortran	Perform the actual comparisons for cmp1.	3.3.4	tem40.a
CNTRL	Fortran	Prepare initial cntrl file from MenuDriver databases	3.3.1	tem40.a
COMP5	Fortran	Compare two Advisor databases node-by-node	3.2.3	tem41.a
comparcp	C shell script	Compare contents of two directories and copy files that differ	3.1.2	
comparem	C shell script	Compare contents of two directories	2.8.8, 3.1.2	nmg.tar
copyem	C shell script	Copy listed files to a directory	3.1.2	
countem	C shell script	Count entries in .pact-files from nmgpact	3.8.4	
EXDOC	Fortran	Extract prologue from a SLATEC-format source code	3.4.1	
EXINFO	Fortran	Extract purpose and GAMS codes from a SLATEC-format source code	3.4.3	
FRED1	Fortran	Reformat label-last file to ref1 form for menu2	3.2.3	tem41.a
HDNG2	Fortran	Prepare headers associated with ingam descriptions	3.3.2	pr10.a
HDNG4	Fortran	Sort descriptions from hdng2 and truncate labels to those from SHOWLB1	3.3.2	pr10.a

<i>Tool Name</i>	<i>Type</i>	<i>Purpose</i>	<i>Reference</i>	<i>Archive</i>
HELPCHK	Fortran	Check out help screens	3.1.3	
insert2	C shell script	Primary subscript for lib1i	3.3.3	insert1.a
lib1i	C shell script	Insert new descriptions into MenuDriver databases	3.3.3	insert1.a
lib1m	C shell script	Create list of GAMS labels from MenuDriver databases	3.3.2	reclass.a
lib1r	C shell script	Prepare MenuDriver insert files from descriptions and GAMS codes	3.3.2	reclass.a
lib1s	C shell script	Delete routines from MenuDriver databases	3.3.1	strip.a
LINECT	Fortran	Eliminate trailing blanks and check character-counts in Advisor database	3.2.3	tem41.a
menu1	C shell script	Create label-last form of menu portion of Advisor or MenuDriver database	3.2	menuf.a
menu2	C shell script	Create new menu portion from label-last form	3.2, 3.3.4	menuf.a
movem	C shell script	Move listed files to a directory	3.1.2	
MTABLE	Fortran	Extract MTABLE contents from MenuDriver databases for lib1m	3.3.2	main.a
nmg.arch	perl script	Periodically save current nmgmail file to archive	3.8.4	
nmgpact	perl script	Compact a file of NMG Usage Statistics	3.8.4	
NTOLE	Fortran	Split MenuDriver database into two parts and prepare tb11	3.3.1	tem40.a
NTOLF	Fortran	Prepare tb12 for inserted routines for lib1i	3.3.3	tem40.a
parse	C program	Parses Fortran programs to determine routines called	3.5.3	
reclass3	C shell script	Primary subscript for lib1r	3.3.2	reclass.a
REF1	Fortran	Reform the menu portion of a database for menu2	3.2.3	tem41.a
runexdoc	C shell script	Run EXDOC on a set of source files	3.4.2	
runexinfo	C shell script	Run EXINFO on a set of source files	3.4.4	
sfg	C shell script	Complete driver for superfile generation. Uses parse and supergen.	3.5.3	
sgfix	C shell script	Fix up results from bssg	3.5.2	
SHEAD	Fortran	Prepare ingam file for strp1	3.3.1	strip.a
SHOWLB1	Fortran	Produce list of non-duplicated GAMS codes for one MenuDriver database	3.3.2	pr10.a
ssg	C shell script	SLATEC superfile generator	3.5.2	
STIME1	Fortran	Split a MenuDriver database into two parts for script menu1	3.2.3	tem41.a

<i>Tool Name</i>	<i>Type</i>	<i>Purpose</i>	<i>Reference</i>	<i>Archive</i>
strp1	C shell script	Prepare ingam file containing deleted descriptions	3.3.1	strip.a
strp2	C shell script	Primary subscript for lib1s	3.3.1	strip.a
supergen	C program	Superfile generator: produces complete superfiles from direct call lists	3.5.2	
T17	Fortran	Put menu portion back in front of file created by T18	3.3.1	tem40.a
T17A	Fortran	Check for consistency of menu nodes with MTABLE in a MenuDriver database	3.3.5	???
T18	Fortran	Construct MTABLE for a set of MenuDriver descriptions	3.3.1	tem40.a
tartoar	C shell script	Convert tar-file to ar-file	2.8.8	nmg.tar
updarf	C shell script	Update an ar-file	2.8.8	nmg.tar
updhsf	C shell script	Update a help screen archive file	3.1.2	

Chapter 4. Validation

4.0. Overview

This chapter describes the tools and procedures that are used to validate the various data files required to support the NMG system. It is assumed that the reader is familiar with the file formats described in Chapter 2 and the relevant maintenance procedures in Chapter 3.

The NMG Validator currently only runs on the LC Cray's. Most of the Fortran programs and scripts it uses are portable, but some details are sure to be platform-dependent.

Contents

4.0. Overview	1
4.1. Why Validate?.....	2
4.1.1. Basic Information	2
4.1.2. Individual Library Validation	2
4.1.3. Overall Validation	4
4.1.4. Short Mathematical Summary of the L-Tests	5
4.2. Running the NMG Validator	7
4.2.0. Background Information	7
4.2.1. Required Files	7
4.2.2. Running a Validation	8
4.2.3. What the Validator Does.....	10
4.2.4. The Validator Report.....	11
4.2.5. The Validator File Archive	13
4.3. How the Validator Works	14
4.3.0. Overview	14
4.3.1. Script Descriptions.....	14
4.3.2. Fortran Programs	18
4.3.3. File Formats	24
4.4. Adding a New Library to the Validator	26
4.4.1. Update Main Scripts	26
4.4.2. Create a Goal File	26
4.4.3. Create Exception Files	26
4.5. Back-up Procedures	28
4.5.1. ar-file Contents	28
4.5.2. SCF Archive Directory Structure.....	28
4.6. Notes on Future Development	30

4.1. Why Validate?

This section sets the scene for description of the NMG Validator by discussing some of the issues that need to be addressed.

4.1.1. Basic Information

Once a database or fetch archive has been created or updated, it is necessary to perform some tests to verify that all of the pieces fit together properly, so that a NMG user will not experience any problems due to improper file formats. The purpose of validation is to answer consistency questions like:

- Does every routine recommended by Advisor appear in at least one MenuDriver database?¹
- Does every routine mentioned by MenuDriver have an on-line document?
- Does every routine mentioned by MenuDriver appear in the appropriate source archive?
- Are the entries in each superfile archive consistent with the associated source and documentation archives?

For reference, we include here a list of the relevant files. Refer to Chapter 2 for information on where to find them.

Table I. File Names for Currently Supported Libraries

<i>Library name</i>	<i>Document archive</i>	<i>Source archive</i>	<i>Superfile archive</i>	<i>Legal notice</i>	<i>Binary library²</i>
MATHLIB	mathdoc	N/A	N/A	N/A	libmath.a ³
MSSL	mssldoc	mssl	superm	legalm	N/A
NAG	nagdoc	N/A	N/A	N/A	libnag.a
PMATH	pmathdoc	pmath	superp	legalp	libpmath.a
SLATEC	sladoc	slatec	supers	legals	libslatec.a
mssl3	N/A	mssl3	super3	legal3	N/A

4.1.2. Individual Library Validation

The following tests should be performed on a library-by-library basis. Each of these includes a reference to the script that performs the associated test. (See Section 4.3.1, where the scripts are described in more detail.)

- L1. Every routine recommended by Advisor from this library must appear in at least one MenuDriver database (with this same library name). (This is list13 consistency check number 5.)

¹ The MenuDriver databases are supposed to describe all user-callable routines. We don't want to recommend something that doesn't exist.

² These are generally in directory /usr/local/lib. (On the Meiko's, most libraries are provided in two forms: lib is replaced by libf for the f77-compiled version or libs for pgf77-compiled.)

³ MATHLIB is available only on the Cray Y-MP, but its documentation is provided by NMG everywhere.

- L2. There must exist a document archive or a superfile archive (or both) for every library. Give an error indication if both are absent. If there is a superfile archive, there must also be an associated source archive. Give an error indication if not. (This is implicitly tested on `lstmake`.)
- L3. Every routine mentioned by MenuDriver must have an on-line document.⁴ Exception: `mssl3` routines do not have documents. Conversely, every entry in the document archive must have a MenuDriver description.⁵ (This is `list13` consistency check number 6.)
- L4. Every routine mentioned by MenuDriver must appear in the appropriate superfile archive (if available).⁶ (This is `list13` consistency check number 4.)
- L5. Every file in the document archive must have a superfile by the same name. [This is to avoid confusion about what to fetch if source is requested.] (Not currently tested.)
- L6. Every file in the source archive must have a superfile by the same name. (This is `list13` consistency check number 7.)
- L7. Any name in a superfile archive that is not in its associated source archive must be in its associated document archive.⁷ (Not currently tested. Handled as a PMATH exception to L9.)
- L8. The first line of a superfile must be the same as the name of the file. Exception: The first line of an entry defined in a C-coded module will be the name of the `.c`-file that defines it. Thus, if the first line of a superfile is not the same as the name of the file, the following two things must be true: (i) the first line ends in `.c`; (ii) there is a file by the name of this superfile in the associated document archive.⁸ (This is tested in `testsup`.)
- L9. Every line of every superfile must be the name of a file in its associated source archive. (This is tested in `testsup`.)
- L10. There must not be any "orphans"; i.e., routines that are not mentioned by MenuDriver and which are not called by any other routine in the library.⁹ (This is tested in `orphgen`.)
- L11. There must exist a legal notice for every library for which source code is available. [We cannot do any further validation beyond checking for its existence.] (Not currently tested. The Fetch Script does check, however.)
- L12. If a binary library is available, there must be an associated document archive. Every routine in the document archive must be in the list of routines in the binary library. Any routine not in the binary library must appear on a known exception list. In the

---

⁴ Routines included in the MenuDriver databases are (by definition) user-callable.

⁵ Otherwise, why is it there? How would a user know to ask for it?

⁶ This used to read "source archive", but the superfile archive is what actually defines what is fetchable.

⁷ The primary example of a superfile name that is not in the source archive is a user-callable entry defined in a C-coded module. Each of these should have an on-line document.

⁸ For now, the fact that `mssl3` has no document archive will not cause problems, because it also has no C-coded modules. What about the future?

⁹ Routines *not* included in the MenuDriver databases are (by definition) subsidiary. These are routines that are in the library only because they are called by some user-callable routine. (There are a few exceptions, such as superseded routines that are retained in SLATEC for compatibility with old applications.)

case of the Cray libraries, this will usually be a subset of `/lib/libsci.a`.¹⁰ (This is `list13` consistency check number 9.)

- L13. If a binary library is available, the list of routines in the binary library must be a subset of those in the superfile archive (if available). Equality is not required, to allow for removal of BLAS, LINPACK, etc. The same exception list as in L12 applies here. Any routine in the superfile archive that is not in the binary library or its exception list must be a `.c-` or `.h-`file name. (This is `list13` consistency check number 1.)

The last two tests are run only if a binary library is available. Tests 4–11,13 are relevant only if source code is available.

Note: Since there are occasionally known exceptions to these basic rules (such as SLATEC "orphans" due to problems with release 4.1 that have been reported to the Committee, or the fact that certain NAG routines do not have individual documents), we maintain exception lists for each library. In the case of tests 12–13, these may be platform-dependent.

4.1.3. Overall Validation

Here we describe tests that are independent of library. (Both types of validations should be performed by the Validator for a complete NMG validation. *Note:* None of the tests specified here are performed by the current Validator.)

- V1. Every library name in an `intree`-file must be in our list of supported libraries. [This is a completion of test L1.] (See Table I for the current list.) This test should be run on both Advisor and MenuDriver databases.
- V2. There must exist an `intree`-file for each label in the main menu, and vice versa. This test should be run on both Advisor and MenuDriver databases. (Currently only tested by visual inspection.)
- V3. Is the tree structure complete and unambiguous? [We first need to determine exactly what this means.] This test should be run on both Advisor and MenuDriver databases.
- V4. Every library mentioned in line 1 of `menu` must actually exist on this platform. [What exactly does it mean for a library to exist? How can this be tested?]
- V5. Is each MenuDriver MTABLE consistent with the associated tree? This means that it should contain no labels that are not contained in the tree and should have no empty nodes. Also, every terminal node in the tree should be in MTABLE.¹¹
- V6. Are there any instances of non-terminal MenuDriver nodes that contain information to be "shown"? The Validator should generate a list of any such nodes and the routines contained in each (with library name). These may be deliberate, or they may indicate

¹⁰ These typically are routines such as BLAS and LINPACK, which we have deleted from the binary version of a library we have installed because we assume that the versions supplied by CRI are more efficient than the Fortran versions. We sometimes also remove routines supplied by `libm.a`.

¹¹ As an alternative, MenuDriver should be modified to properly deal with empty nodes. (This would allow us to include the complete GAMS classification, even though there are categories for which none of our current libraries supply routines.)

where a refinement of the GAMS classification is needed. [The deliberate ones could be an exception list for an automated validation.]

- V7. Each file mentioned in the list-files in `$nmgdir/help` must actually exist in directory `$nmgdir/help/helpfiles`.¹² Conversely, each file in this directory (except `helpmenu`) must be mentioned in at least one list-file.

We would like to insure the completeness of the Library Availability screen, for example, but there is probably no other mechanical validation that can be performed on the helpfiles.

4.1.4. Short Mathematical Summary of the L-Tests

For completeness, we provide here brief mathematical definitions for the tests described in Section 4.1.2. In the following,

`adv` is the list of routines from this library recommended by Advisor,
`mdr` is the list of routines from this library which have MenuDriver descriptions,
`doc` is the list of files in the document archive for this library,
`src` is the list of files in the source archive for this library,
`super` is the list of files in the superfile archive for this library,
`legal` is the name of the legal notice for this library,
`bin` is the list of entries in the precompiled binary for this library,
`except` is the binary exception list (routines explicitly removed from `bin`),
`name(superfile)` is the name of file `superfile`,
`superfile[n]` is line `n` of file `superfile`,
`|superfile|` is the number of lines in file `superfile`, and
`{x} = {name | name:e = x}`, where `" : e"` is the C shell notation for "extension".
 (For example, if `name=base.c`, `name:e = c`.)

- L1. `adv` is contained in `mdr`.
 L2. `doc` or `super` exists; if `super` exists, then `src` exists.¹³
 L3. `mdr` equals `doc`.
 L4. `mdr` is contained in `super`.
 L5. `doc` is contained in `super`. [If L3 is true, L4 and L5 are equivalent. Since `mss13` has no `doc`, L4 is the test of choice. If L3 is false, however, *both* should be tested.]
 L6. `src` is contained in `super`. [For strictly Fortran libraries, this should be "equals".]
 L7. `super - src` is contained in `doc`. [Empty for strictly Fortran libraries.]

¹² `nmgdir` denotes a script variable that points to the "public" version of NMG. On UNICOS and milstein, `nmgdir = /usr/local/math/nmg`.

¹³ `mss13` is the only case where `super` exists but `doc` does not.

- L8. For each superfile contained in super,
    either (superfile[1]=**name**(superfile)),
    or (superfile[1]:e=c *and* **name**(superfile) is contained in doc).
- L9. For each superfile contained in super,
    { superfile[n] : n= 1, ..., |superfile| } is contained in src.
- L10. (The orphan test)
    For each superfile contained in super,
        if (**name**(superfile) is *not* contained in mdr),
        then (**name**(superfile) is in some other superfile in super).
- L11. If super exists, then legal exists.
- L12. If bin exists, then doc exists
    and doc is contained in bin - except.
- L13. If bin exists and super exists, then
    super-{c}-{h} equals bin - except.

4.2. Running the NMG Validator

This section describes how to run the NMG Validator and how to interpret its output.

4.2.0. Background Information

Validator checks for consistency among component parts of a given library. It runs most of the tests discussed in Section 4.1.2. Currently, the libraries are IMSL, MATHLIB, MSSL, mssl3, NAG, PMATH, and SLATEC.¹ The component parts and their abbreviations are shown below:

adv	Advisor file list: each line contains a routine name, the library name, and an Advisor label, as extracted from the current Advisor databases.
blib	binary file list: each line contains a compiled routine name from the binary version of this library. ²
craylib	Cray SCILIB list: each line contains the name of an entry in libsci.a, truncated to eight characters. ³
doc	document file list: each line contains the name of a document from the documentation archive for this library.
mdr	MenuDriver file list: each line contains a routine name, the library name, and a GAMS classification, as extracted from the current MenuDriver databases.
src	source file list: each line contains the name of a routine for which source code is available in the source archive for this library.
sup	superfile list: each line contains the name of a superfile from the superfile archive for this library.

4.2.1. Required Files

The following files must be present in the current working directory in order for a validation to be run:

```
dup1
show1
show2
show4
show5
show7
showa
startval
liblist
val.a
```

¹ IMSL and MATHLIB are on the Cray Y-MP only.

² Clearly relevant only where a binary library is available.

³ This list is generated because we have deleted BLAS, LINPACK, and other routines from several of the binary libraries installed on the Cray's, under the assumption that their SCILIB equivalents have been optimized for best performance on this machine. We check that only these routines are listed when we compare blib with src.

vallib4.a

val.a and vallib4.a are ar-format archive files. val.a contains Fortran programs used by the Validator. vallib4.a contains scripts used during validation, goal files for inclusion tests to meet, exception files, and a README file.

startval is the starter script for Validator, and liblist is a file that tells startval what libraries can be validated on this platform. The first seven files listed above are executables of Fortran codes whose source files are stored in val.a. If these are not present, startval will compile them for use during the run. To save compile time, in case the user wishes to do more validation, these are not destroyed after the run.

All of these files will be found in the current val archive (see Section 4.5).

4.2.2. Running a Validation

To start the Validator, simply type "startval". A typical validation session, in which we decide to validate PMATH, is reproduced below. As usual, characters typed by the user are **boldface**.

sn1007% **startval**

Begin Validator

(A code compilation message could be here.)

Please enter your initials

rpd

(menu produced by script startval)

```
1  IMSL
2  MATHLIB
3  MSSL
4  NAG
5  PMATH
6  SLATEC
7  mssl3
```

Do you want to run all or some libraries? (a,s)

s

Enter list of numbers for libraries to be run
in the range 1 to 7 separated by spaces.

5

Libraries 5 selected

Do you want to proceed, quit, or start again? (p,q,s)

p

Running libraries

PMATH:

list maker:

(from script lstmake)

duplicates out:

(from script dupper)

super file tests:

(from script testsup)

inclusion tests:

(from script list13)

orphan tests:

(from orphgen)

End validator

The response to the first question is passed on to various scripts to include identification of the person running the validation in the output. After the menu display, the user is given a choice of validating all libraries or only a subset. Here we have elected to validate just PMATH (item 5 in the menu). The final question is to allow correction of the list of libraries before the actual validation run starts. The various scripts mentioned above will be described in subsequent sections.

After the validation has run, the following additional files will be present:

`mdrivelst` contains the complete list of MenuDriver names (see Section 4.3.1).⁴
`v_report` contains the validation report.
`vfiles.lib` is an ar-file that contains additional support information produced by the Validator.⁵

The following error messages might occur during a validation (generated via echo):

From script `lstmake`:

```
Error, blib is empty.
Error, craylib is empty.
Error, src is empty.
Error, adv is empty.
Error, doc is empty.
Error, mdr is empty.
Error, sup is empty.
```

From script `orphgen`:

```
ERROR, orphans empty, orphan list not empty
ERROR, orphans exist, orphan list empty
ERROR, orphan exception difference
```

From script `exceptest`:

```
$1 "ERROR exception difference with " $nam1 " \ " $nam2
$1 "ERROR exception middle with " $nam1 " , " $nam2
$1 "ERROR exception difference with " $nam2 " \ " $nam1
```

From script `goval`:

```
Error, invalid inclusions
```

The `exceptest` script variable 1 will contain a test number; variables `nam1` and `nam2` will contain the names of the files involved in the erroneous test. These will be found in `vfiles.lib`.

If any of these error messages appears at the terminal, the user should carefully examine `v_report` for more information on what has gone wrong. It will generally be necessary

---

⁴ This is useful only for comparison with a `mdrivelst` from a previous validation to help confirm that a set of MenuDriver database updates has done what was expected.

⁵ The suffix will be the name of the library whose validation has been run. In this case, the file will be named `vfiles.PMATH`.

to examine one or more of the `vfiles.lib` files to determine the exact cause of the error(s).

Before displaying the contents of the files generated during the PMATH validation, it will be necessary to discuss the various validation tests in a bit more detail.

4.2.3. What the Validator Does

The NMG Validator performs most of its tests by generating lists of names, computing symmetric differences between various lists, and verifying that desired relationships exist among the components of the symmetric differences.

One of the main scripts of Validator, `lstmake`, generates the lists of names of routines mentioned in Section 4.2.0. For certain libraries, some of the component parts are empty, while for some libraries all seven components are present. For example, MSSL does not have a binary component, NAG does not have a source component and consequently does not have a superfile component either, SLATEC and PMATH have all components.

One of the first consistency checks for Validator is done in `lstmake`. `lstmake` contains tables of which libraries have which components. If `lstmake` generates an empty file for a library where that component is not supposed to be empty, then an error is signaled.

In order to validate a new set of Advisor or MenuDriver databases before they are moved to the public directories built into `lstmake`, the script checks for the existence of files `new-advisor` or `newmdrive`. If either exists, it is assumed to be an ar-file containing a new set of intree-files to be validated, and the associated list is extracted from these.

Further consistency checks occur in script `testsup` (test superfiles). For example, the first name in any superfile must be the name of the superfile itself or must end in `.c`. There is also a second consistency check in `testsup`, and this will be described later. Naturally, any time an inconsistency is found it is reported as an error.

Up to nine consistency checks are performed by script `list13` and by a subscript of `list13` called `exceptest`. The number on the left side of a consistency check report (see next subsection for an example) refers to the number of the inclusion test. Any component mentioned in one of these tests is nonempty. Using the tool of symmetric differences, pairs of library components are examined to see if certain inclusion tests are satisfied. `list13` checks universal goals. For example, the universal goal for test 6 is "mdr equals doc".

Sometimes the universal goal is not attainable, and exceptions occur with the knowledge of the librarian. Lists describing these exceptions are stored in Validator's `efiles` (exception files) file for this library. If an `efiles` entry for a check is present, then the universal goal is modified to reflect the exception(s). For example, the PMATH goal for test 6 is "mdr equals doc within exception left", which means that `mdr-doc`, rather than being empty, must match a known exception list. These exception tests are performed in

exceptest. Of course, the names in the exception files must be the only exceptions, or an inconsistency is reported.

The inclusions found must match the expected inclusions for the library, where the expected inclusions are stored in files with names `goallib`, where `lib` is the library name. (See Section 4.3.3 for more details.)

Validator consists of several scripts, Fortran codes, and expectation files (`goal`-files for inclusion, and `efiles`-files for exceptions). The Fortran codes are stored in ar-file `val.a`. The scripts, except for starter script `startval`, and the expectation files are stored in ar-file `vallib4.a`. These are all described in Section 4.3.

4.2.4. The Validator Report

The Validator report, `v_report`, is present at the end of a Validator run. It represents a summary of three Validator tests: inclusion, superfile, and orphans. It is good practice to examine `v_report`, even if no error messages have appeared during the validation. The output for library `PATH` from the Validator run shown above is used as an example. For this report, explanatory comments have been inserted after each of the three parts.

Thu May 11 15:21:20 PDT 1995 sn1007

*****

PMATH inclusion comparisons:

```
1: blib equals sup within exception left
3: blib is disjoint from craylib
4: mdr is contained in sup within exception left
5: adv is contained in mdr
6: mdr equals doc within exception left
7: sup equals src within exception left
8: doc is contained in blib
```

Comments on inclusion tests: Nine tests are possible at this point (see Section 4.3.1 for the complete list). Tests 2 and 9 involve `sup-blib`. These tests are missing here because `sup equals blib` for `PMATH`, and therefore `sup-blib` is empty. Tests are not run using empty input files.

The above inclusion tests were run by script `list13`, and their output agrees with file `goalpmath`. An error would be signaled if disagreement had occurred. The agreement test is performed in script `goval` using UNIX utility `diff`. Notice that exception files exist for `PMATH`. In this case all inclusion tests have passed. Error messages would appear here to indicate failure(s).

PMATH super file tests:

Warning, cv16to64 has `pmath_cnv.c` as first line

```
Warning, cv64to16 has pmath_cnv.c as first line
Warning, ranf8 has pmath_rnf.c as first line
Warning, ranf9 has pmath_rlf.c as first line
Warning, rlfcnt has pmath_rlf.c as first line
Warning, rlmset has pmath_rlf.c as first line
Warning, rlsset has pmath_rlf.c as first line
Warning, rlsset has pmath_rlf.c as first line
Warning, rnfcnt has pmath_rnf.c as first line
Warning, rnmset has pmath_rnf.c as first line
Warning, rnsget has pmath_rnf.c as first line
Warning, rnsset has pmath_rnf.c as first line
```

First name of super must match name of super or end in .c

```
No. of good files: 91
No. of cfiles    : 12
No. of bad files : 0
```

```
79      79      565  nodups
79      79      565  src
```

diffsup is the diff of src and nodups.
 nodups is the union of the super files with
 duplicates removed.
 Note: diffsup should be empty.

```
0      0      0  diffsup
```

Two tests are run with respect to the superfiles by script testsup.

- (1) Test that the first name of each superfile is the name of that superfile, or the name ends with .c. The first few lines, through the "No. of bad files" line, report the results of test (1). The number on this line must be zero for a successful test. The number of files where the first name ends in .c is reported as "No. of cfiles".⁶
- (2) Test that every name in each of the superfiles is a name in source file list. The result of this test is a file diffsup, which should be empty. The output of "wc diffsup" is printed as the last line, and it should contain all zeroes.

PMATH orphan test passed

Comments on orphan testing: By definition, an orphan is a routine not called by any other routine which is not mentioned in any MenuDriver database (list mdr). A list of programs not called by other programs is constructed from the library's superfiles and saved by Validator as file single.⁷ Script orphgen generates a list of orphans for this library by comparing single with mdr. It then compares that with a known list of orphans, if one exists. The simple output reproduced above indicates that no error condition has occurred

---

⁶ Note: At this time the only library with .c endings is PMATH. The "Warning" messages are merely printed to call attention to those superfiles that do not fit the standard pattern. In this case, these are user-callable entries defined in the indicated .c-files.

⁷ Without source there are no superfiles, and hence orphan testing would be meaningless.

for PMATH. Had there been errors, more details would have appeared in the orphan report.

The scripts mentioned above are described Section 4.3.1.

4.2.5. The Validator File Archive

A Validator file archive, `vfiles.lib`, for each library `lib` tested, is present at the end of a Validator run. It contains files that show the details behind the test results that are reported in `v_report`.

For our example run, `vfiles.PMATH` contains:

`d_report` – a summary of the symmetric differences computed during the validation.

`dfiles.PMATH` – an ar-file containing the detailed symmetric difference output files. These have names of the form `diff1`, where `1` is the number of the associated test. It also contains file `duplst`, which contains names removed from various lists because of duplication.

`lfiles.PMATH` – an ar-file containing all of the list-files mentioned in 4.2.0, as well as `single` and `orphan_out`. The latter is the orphan report, which is also part of `v_report`.

`orphan_out` – another copy of the orphan report.

`sdmdr` contains the single/double names from the MenuDriver databases. The result of "head sdmdr" illustrates its contents:⁸

BINOM	DBINOM	IMSL	C1
CBRT	DCBRT	IMSL	C2
HYPOT	DHYPOT	IMSL	C2
CSEVL	DCSEVL	IMSL	C3
ACOSH	DACOSH	IMSL	C4a
ASINH	DASINH	IMSL	C4a
ATANH	DATANH	IMSL	C4a
COSDG	DCOSDG	IMSL	C4a
COT	DCOT	IMSL	C4a
SINDG	DSINDG	IMSL	C4a

`sfiles.PMATH` – an ar-file containing the lists of names produced by symmetric differences. Each symmetric difference produces three lists: `d1`, `d2`, and `d3`. In the ar-file, these have names of the form `sym1-dn`, where `1` is the number of the associated test, and `n` is 1, 2, or 3. For example, `sym1-d2` is `d2` of `diff1`. In the inclusion part of the Validator report these are referred to as the "left", "middle", and "right" parts, respectively. The parts must agree with exception files, when present, or an error is signaled.

⁸ This list was generated primarily because the documentation archive for IMSL contains only single precision names, so the double precision names constitute a known exception list for test 6. If IMSL is eventually removed, generation of this file can probably be eliminated.

4.3. How the Validator Works

This section describes how the NMG Validator actually performs the tests described in the previous section.

4.3.0. Overview

Validator consists of ten C shell scripts. Following is a list of the scripts, with indentation showing level of control. The scripts are called in the indicated order.

```
startval
  startval4
    goval
      lstmake
      dupper
      testsup
      list13
      exceptest
      orphgen
    mclean
```

Following are brief descriptions of the purposes of these scripts. All but startval are stored in ar-file vallib4.a.

```
startval  - overall Validator control script.
startval4 - validate a particular library.
goval     - "main code" script for startval4.
lstmake   - generate library lists.
dupper    - eliminate duplicates from various lists of names.
testsup   - run superfile tests.
list13    - run inclusion tests.
exceptest - run exception file tests.
orphgen   - run orphan test.
mclean    - clean up after validation.
```

4.3.1. Script Descriptions

Almost all scripts have arguments. Following is a list with calls showing sample arguments to the right:

```
startval
startval4  mss13  rpd
goval     mss13  rpd
lstmake   mss13  rpd
dupper
testsup
```

```
list13      mss13   rpd
exceptest   blib    sup   exc11      exc12      exc13      1:
orphgen     mss13   rpd   orphlist
```

Files `exc11`, `exc12`, ... `orphlist` are stored in ar-files `efiles.lib`, where `lib` at present has four possible values: `IMSL`, `NAG`, `PMATH`, or `SLATEC`. (`efiles` stands for "exception files".) `exc1n` is the exception file corresponding to `sym1-dn` (see Section 4.2.5). At present only `efiles.SLATEC` has a file `orphlist`.

Each of these scripts and input files will be described in more detail below.

startval. This is the main control script for the NMG Validator. It performs the following steps:

1. Compile worker Fortran codes from `val.a`, if not already present. (See Section 4.3.2. for information on the Fortran codes.)
2. Display menu for validation, and ask user for choice. The menu is constructed from file `liblist`, which simply contains a list of the libraries for which validations make sense on this platform. (This, of course, must be a subset of the libraries that `startval` already knows about.) One can validate all libraries or one can validate a specified subset.
3. Invoke `startval4`, which validates a particular library. In the case of validating all libraries, this is done in a loop over all possible libraries.
4. Finish Validator report and clean up. In the case of validating all libraries, this stage concatenates all library Validator reports into a single report, `v_report`.
5. Quit

startval4. This script validates a particular library. It is invoked with two arguments, the name of the library and the initials of the person running the validation (which are passed on to various subsidiary scripts and codes for identification purposes).

`startval4` first extracts the scripts used for validation from `vallib4.a`. It then invokes `goval`, which calls other scripts in the proper order. It prepares the validation report for the library being validated and then deletes unnecessary files.

goval. This script actually controls most of the work of a validation. Like `startval4`, it is invoked with two arguments, the name of the library and the initials of the person running the validation.

As noted above, `goval` invokes the scripts `lstmake`, `dupper`, `testsup`, and `list13` in succession to create the necessary lists and perform validation tests based on them. The output from the inclusion tests run by `list13` is written to file `diffout1`, which is compared with the `goal`-file for this library. It concatenates the output generated by the various tests and cleans up via script `mclean`.

lstmake. This script generates library lists. Like `startval4`, it is invoked with two arguments, the name of the library and the initials of the person running the validation.

`lstmake` sets up paths to various library components, then extracts and constructs lists from these components. This is perhaps the most crucial part of Validator, since all subsequent tests are based on these lists. As noted in Section 4.2, the possible components are `adv`, `blib`, `craylib`, `doc`, `mdr`, `src`, and `sup`. For a particular library, some of these components may be empty, but `lstmake` signals an error if an expected component is missing.

Before constructing `adv` or `mdr`, `lstmake` checks for the existence of files `newadvisor` or `newmdrive`. If either exists, it is assumed to be an `ar`-file containing a new set of `intree`-files to be validated, and the associated list is extracted from these, rather than the "public" copies. To create `mdr`, `lstmake` first creates `mdrive1st`, a complete table of routine names, libraries, and GAMS codes for all descriptions in all MenuDriver databases. List `mdr` for a particular library is created via `"grep lib mdrive1st"`. File `mdrive1st` is left on disk after the validation and archived in the `results`-file for comparison with one from a previous validation.

If exception files exist for a library, they are extracted from `efiles.lib` within `lstmake`. UNICOS utility `bld` is used to partially construct lists `blib` and `craylib`.¹ Lists are not complete until duplicates have been removed, the proper case is used, and sorting has taken place. Other types of massaging of these lists also takes place in `lstmake`.

dupper. This is a minor script that eliminates duplicates from various lists of names generated during the validation. Although it has an important function, `dupper` is not called a main script, because the main scripts have *many* functions to perform.

testsup. This script runs the superfile tests. `testsup` is run only if `sup` exists.²

As noted in Section 4.2.4, this script runs two tests on the superfile archive `super`:

1. Test that the first name in each superfile in `super` is the name of that superfile, or the name ends with `.c`.
2. Test that every name in every superfile is a name in the source file list `src`. To accomplish this, all of the superfiles in `super` are merged into a file, sorted, and a file without duplicate names, called `nodups`, is created. This file is compared with `src` via `"diff nodups src > diffsup"`, and the output of `"wc diffsup"` is printed, to verify that it is empty.

list13. This script runs the inclusion tests that make up the bulk of the validation. Like `startval4`, it is invoked with two arguments, the name of the library and the initials of the person running the validation.

This script performs symmetric difference calculations on paired components of the library.³ The subsets (sublists) of each symmetric difference are stored, and later are used by script `exceptest` if exception files exist.

¹ This is the part of Validator that is most likely to need changing when porting to another platform.

² Note: `src` and `sup` always exist or do not exist together.

³ Here "component" refers to one of the lists generated by `lstmake`.

There are nine main symmetric difference tests run by `list13`:

- 1: `blib` vs. `sup`⁴ (calculates `sup-blib`)
- 2: `sup-blib` vs. `craylib`
- 3: `blib` vs. `craylib`
- 4: `mdr` vs. `sup`
- 5: `adv` vs. `mdr`
- 6: `mdr` vs. `doc`
- 7: `sup` vs. `src`
- 8: `doc` vs. `blib` (calculates `doc-blib`)
- 9: `doc-blib` vs. `sup-blib`

The complete output of each of these tests is stored in `diff`-files, `diff1` through `diff9`. These are considered supporting validation information files, and are saved in `ar-file dfiles.lib`. The associated sublists are saved in `ar-file sfiles.lib`. Note that a symmetric difference test is not run if one of the paired components is empty.

A complete output log is written to `diffout`. A summary of the output from the inclusion tests is written to file `diffout1` for comparison with the goal file for the particular library being validated. `list13` also invokes `orphgen`, which checks for orphans.

exceptest. If exception files exist, this script is executed by `list13` to perform exception file testing. It is invoked with six arguments, as follows:

```
exceptest list1 list2 excl1 excl2 excl3 l:
```

where `l` is a test number, `list1` and `list2` are the two lists involved in test `l`.

It is assumed that the symmetric difference files `d1`, `d2`, `d3` are present and that these were computed from `list1` and `list2`. The `exc`-files, if they exist, are the exception files for the left, middle, or right part of the symmetric difference, respectively. If the `n`-th `exc`-file exists, it is compared with `dn` and an error reported if either `dn` is empty or the files do not agree.

orphgen. This script runs the orphan test. It is invoked with three arguments. The first two are as for `startval4`; the third is `orphlist`. `orphgen` is only run if `sup` exists.

The script will calculate a list of orphans if they exist. Orphans are defined as follows:

- (1) The set of all library programs not called by another library program might be called "singles". If they exist, `orphgen` calculates a list of their names, and this list is called `single`. The list is produced by Fortran code `DUP1`,⁵ which eliminates duplicates from a sorted list. `DUP1` is called first by script `dupper` and last by script

⁴ The list that is used for `sup` in test 1 is actually `sup-c-h`, which is `sup` with any names that end in `.c` or `.h` removed. Thus, the file called `sup-blib` is actually `(sup-c-h)-blib`. The true `sup` is used in tests 4 and 7, however. (This distinction is currently only relevant for `PMATH`.)

⁵ See Section 4.3.2 for more information on this and other Fortran codes used by these scripts.

`testsup`, where it was working on a list of *all* names appearing in superfiles. A "single" appears only once in this list, which means that its name appears only in its trivial superfile (or else a typographical error has occurred in creating some superfile).

(2) Calculate all library programs not mentioned by MenuDriver (`mdr`).

Orphans are the names in common to `single`, from step (1), and the list of names produced by step (2).

Ideally, a library has no orphans. Sometimes a library has orphans which the librarian elects to leave alone for some special purpose. These are placed in a special list called `orphlist`, which is annotated and stored in the `efiles` (exception files) archive for that library. If known orphans exist in an exception file, then exception testing takes place in `orphgen`. There are three possible error conditions:

- a. orphans exist but there is no known orphan list;
- b. orphans do not exist and a known orphan list does exist;
- c. both lists exist, but are not equal.

If no error occurs, the script reports that the orphan test has passed. In any event, the output from `orphgen` is written to file `orphan_out`.

mclean. This is a minor script which cleans up after a validation. "Cleaning up" means putting away supporting files of information generated by the validation, and removing unwanted files after validation. The user may obtain the supporting files at the end of a validation by accessing ar-file `vfiles.lib`.

4.3.2. Fortran Programs

As mentioned in Section 4.2.1, there are seven Fortran programs that are used by the above scripts to do part of their work. Their source codes are stored in ar-file `val.a`. Following is a list giving the code-source file correspondence and a brief purpose for each.

DUP1 (source code: `dupnamb.f`) – remove duplicates.

SHOWA (source code: `showta.f`) – extract information from Advisor databases.

SHOW1 (source code: `showt1a.f`) – massage SHOWA output into a more usable form.

SHOW2 (source code: `showt2b.f`) – extract information from MenuDriver databases.

SHOW4 (source code: `showt4b.f`) – split a name from right-side of colon separator.

SHOW5 (source code: `showt5.f`) – strip off certain leading characters.

SHOW7 (source code: `showt7c.f`) – compute the symmetric difference of two files.

Each of these programs will be described in more detail below.

Caution: While these codes are all basically standard Fortran 77, they contain a few known Cray-specific library calls. These are routines like `DATE` and `CLOCK` that are used to identify the output. Also, the `CALL EXIT` statements should be replaced by `STOP` throughout.

DUP1. This code removes duplicated lines from a sorted file. It reads 80-character lines, but only compares the first 8 characters, which are presumed to be names. DUP1 originally ran interactively and prompted for its input, but it is currently used as follows:

```
dup1 < dupin
```

where *dupin* has three lines as follows:

```
INPUT   -- the name of the input file (8 characters maximum);
OUTPUT  -- the name of the output file (8 characters maximum);
NAME    -- user's name or initials (10 characters maximum) .
```

INPUT is the name of the file from which duplicates are to be removed. The result without duplicates will be written as file *nodups* (no choice of name). OUTPUT is actually a log file, containing detailed information as to where duplicates were found.

This code also produces three other output files as by-products:

```
single  the list of "singles"; i.e., lines that appeared only once in INPUT.
outdup1 A file which will be empty if there were no duplicates or will contain a single
        line if any duplicates were found.
outdup2 A file that will be empty if there were no duplicates or will contain a single
        line telling how many duplicates were found.
```

SHOWA. This code extracts information from Advisor databases for *lstmake*. It is used in the creation of *list adv*.

SHOWA assumes the existence of files *namedata*, *intree1*, ..., *intree8* and extracts program names, the library to which they belong, and their decision tree label from all of the "Use" blocks of the terminal nodes. *namedata* simply contains a single line with the initials of the user, for identification purposes. The output is written to file *lfirst* (label first).

To illustrate what SHOWA does, here is a portion of *intree1*:

```
First kind                                1e1
  Use RFL (MSSL3) .
Second kind                               1e2
  Use RDL (MSSL3) .
Third kind                                1e3
  Use RJL (MSSL3) .
Sine Integral                             1f1
  Use S13ADF (NAG) .
Cosine Integral                           1f2
  Use S13ACF (NAG) .
E1                                         1g1
  Use E1 (SLATEC) .
EI                                         1g2
  Use EI (SLATEC) .
E(n+k,x)                                  1g3
  Use EXINT (SLATEC) .
```

and here is the associated portion of *lfirst*:

```
t 1e1
    Use RFL (MSSL3) .
t 1e2
    Use RDL (MSSL3) .
t 1e3
    Use RJL (MSSL3) .
```

```

t 1f1
    Use S13ADF (NAG) .
t 1f2
    Use S13ACF (NAG) .
t 1g1
    Use E1 (SLATEC) .
t 1g2
    Use EI (SLATEC) .
t 1g3
    Use EXINT (SLATEC) .

```

Note that the first part of `1first` contains the header information from the `intree`-files that were processed by `SHOWA`, as well as the date and time of the run and the `namedata` information.

SHOW1. This code is used to massage the `1first` file created by `SHOWA` into a more usable format. It also expects `namedata` as an input file. Its output is `1comp`.

Here is the relevant portion of `1comp` produced by `SHOW1` from the above `1first` file:

RFL	mssl3	1e1
RDL	mssl3	1e2
RJL	mssl3	1e3
S13ADF	NAG	1f1
S13ACF	NAG	1f2
E1	SLATEC	1g1
EI	SLATEC	1g2
EXINT	SLATEC	1g3

Note that `1stmake` extracts the `adv` information for a particular library via a "`grep lib 1comp`" command and converts the result to lower case.

SHOW2. This code extracts information from `MenuDriver` databases for `1stmake`.

`SHOW2` assumes the existence of files `intreeA`, ..., `intreeZ` and extracts program names, the library to which they belong, and their GAMS classification codes. It is used as

```
show2 < namedata
```

where `namedata` is as for `SHOWA`. There are two primary output files, `out1` and `out2`. The first is basically a log file, containing the identification lines from the `intree`-files. The actual output appears in `out2`.

To illustrate its contents and format, here is the first part of `out2`, corresponding to the file `intreeA` displayed in Section 2.5.2.

F06BLE	NAG	A3
XADD	SLATEC	A3
DXADD	SLATEC	A3
XADJ	SLATEC	A3
DXADJ	SLATEC	A3
XC210	SLATEC	A3
DXC210	SLATEC	A3
XCON	SLATEC	A3
DXCON	SLATEC	A3
XRED	SLATEC	A3
DXRED	SLATEC	A3
XSET	SLATEC	A3

DXSET	SLATEC	A3
BRENT	mss13	A3
EXRPACK	mss13	A3
A02ABE	NAG	A4
A02ACE	NAG	A4
C06GBE	NAG	A4
C06GCE	NAG	A4
C06GQE	NAG	A4
C06GSE	NAG	A4
F06CLE	NAG	A4
CARG	SLATEC	A4
R9PAK	SLATEC	A6
D9PAK	SLATEC	A6
R9UPAK	SLATEC	A6
D9UPAK	SLATEC	A6
C06BAE	NAG	A7
C06GBE	NAG	A7
C06GCE	NAG	A7
C06GQE	NAG	A7
C06GSE	NAG	A7

Note that `1stmake` extracts the `mdr` information for a particular library via a "`grep lib out2`" command and converts the result to lower case.

As a spin-off, `SHOW2` produces file `smdmr`, containing the single/double names found in the databases. (See Section 4.2.5 for an example.) This is currently only useful for `IMSL`.

SHOW4. This code was originally used to process "`bld t`" output from binary libraries, such as `libsci.a`, which had entries of the form "`file.o:NAME`".⁶ It produces the name following the colon in all such lines; if a line contains no colon, `SHOW4` simply produces the name. *Caution:* `SHOW4` assumes that names have at most eight characters. All names in `MSS`-supported libraries satisfy this requirement.

For example, if the input file `cray1` contained the following lines,

```
setnproc@.o:SETNPROC@
caxpy.o:CAXPY
CCOPY
cdotc.o:CDOTC
cdotu.o:CDOTU
cgbmv.o:CGBMV
cgbmv1@.o:CGBMV1@
```

the output `cray2` would have:

```
SETNPROC
CAXPY
CCOPY
CDOTC
CDOTU
CGBMV
CGBMV1@
```

Note that since `SHOW4` picks up at most eight characters after the colon, the trailing "@" has been lost from the first name.

---

⁶ Since we are now using "`bld te`" to get entries, this code may no longer be useful.

SHOW5. This code is called after SHOW4 to strip away certain leading characters (namely `_`, `%`, `$`) that the Cray compiler `cft77` appends to certain entry names. It reads `cray2` and produces `cray3`.

For example, the following is a partial "diff `cray2` `cray3`" output:

```
< $SCERP
---
> SCERP

< _qccopy2
< _qccopy2
< _qcgemm$
< _qcswap$
---
> qccopy2
> qccopy2
> qcgemm$
> qcswap$
```

SHOW7. This code, which computes the symmetric difference of two files (lists), is the workhorse of script `list13`. It is used as

```
show7 < in7
```

where `in7` has four lines as follows:

```
INPUT1  -- the name of the first input file (12 characters maximum);
INPUT2  -- the name of the second input file (12 characters maximum);
OUTPUT  -- the name of the output file (12 characters maximum);
NAME    -- user's name or initials (10 characters maximum) .
```

INPUT1 and INPUT2 are the names of the files whose symmetric difference is to be computed. The result will be written to files `d1`, `d2`, `d3`. Here `d1` contains names in INPUT1 but not in INPUT2 (the "left" portion), `d2` contains names in common to both files (the "middle" portion, or intersection), and `d3` contains names in INPUT2 but not in INPUT1 (the "right" portion). **Caution:** In order for SHOW7 to work properly, both input files must be sorted.

OUTPUT contains detailed information about the two input files, showing how many entries are in each of the three portions and printing up to 1000 names from each. SHOW7 produces two additional output files. `out7` is a brief summary of the information in OUTPUT. `out7a` contains a single line, suitable for inclusion in a goal file.

Figure 1. Symmetric Difference Components

To illustrate the use of SHOW7, here are two simple lists:

```

::::::::::::::::::
SanDiego_Zoo
::::::::::::::::::
bear
elephant
giraffe
koala
lion
tiger
zebra

::::::::::::::::::
SanFran_Zoo
::::::::::::::::::
bear
cheetah
elephant
lion
tiger
warthog

```

We executed "show7 < in7ex", where the in7ex file is:

```

SanDiego_Zoo
SanFran_Zoo
out7ex
FNF

```

The outputs were as follows:

```

::::::::::::::::::
out7
::::::::::::::::::

SanDiego_Zoo has 7 names.
SanFran_Zoo has 6 names.
Items in SanDiego_Zoo not in SanFran_Zoo : 3
Items common to SanDiego_Zoo and SanFran_Zoo : 4
Items in SanFran_Zoo not in SanDiego_Zoo: 2
-----

::::::::::::::::::
out7a
::::::::::::::::::
neither SanDiego_Zoo nor SanFran_Zoo is contained in the other.
::::::::::::::::::
out7ex
::::::::::::::::::
showt7b                                3/29/96    11:37    FNF
input file one: SanDiego_Zoo
input file two: SanFran_Zoo

SanDiego_Zoo has 7 names.
SanFran_Zoo has 6 names.
Items in SanDiego_Zoo not in SanFran_Zoo : 3
Items common to SanDiego_Zoo and SanFran_Zoo : 4
Items in SanFran_Zoo not in SanDiego_Zoo: 2

Items in SanDiego_Zoo not in SanFran_Zoo : 3
giraffe      koala      zebra

Items in SanFran_Zoo not in SanDiego_Zoo: 2
cheetah      warthog

Items common to SanDiego_Zoo and SanFran_Zoo : 4

```


bear	elephant	lion	tiger

The d-files were printed in out7ex, in the order 1, 3, 2. For completeness, here they are:

:::::::::::::	:::::::::::::	:::::::::::::
d1	d2	d3
:::::::::::::	:::::::::::::	:::::::::::::
giraffe	bear	cheetah
koala	elephant	warthog
zebra	lion	
	tiger	

4.3.3. File Formats

We will not attempt to give the detailed formats of all the files that appeared in the above descriptions. We will discuss the three main types of files, however.

List Files. The various list files that are used in inclusion testing are merely lists of names, one name per line. In order for comparisons and symmetric differences to perform correctly, the names are all lower case, the lists have been sorted, and duplicates have been removed. Many of the tools assume that a name has a maximum of eight characters, which might cause problems in the future as we include more C-coded modules.

Exception Files. Each file that is stored in `efiles.lib` is basically just a sublist of one of the list files. These contain lists of names that are known exceptions to some ideal goal for one of the inclusion tests. In order for comparisons to perform correctly, the names must be all lower case and in sorted order.

Since only the first ten characters are read from an exception file, the remainder of the line can contain a brief explanation for why this name is an exception to the rule. We have adopted the standard of putting a colon in column 11 to separate the name from the annotation. For example, the following is the `orphlist` file for SLATEC:

```

cfft b      :Deprecated routine; superseded by cfft b1.
cfft f      :Deprecated routine; superseded by cfft f1.
cfft i      :Deprecated routine; superseded by cfft i1.
cpevlr      :Glitch, subsidiary of cpzero? (old vers. of cpev?)
dlmerg      :Complete the set of routines slmerg and clmerg, FISHPACK.
d9atn1      :Double precision version of r9atn1.
d9ln2r      :Double precision version of r9ln2r.
dfulmt      :Co-routine, see READ.exception
drsc o      :Co-routine.
dsvc o      :Co-routine.
fulmat      :Co-routine.
ilmerg      :Complete the set of routines slmerg and clmerg, FISHPACK.
rsc o      :Co-routine.
sods        :Glitch, subsidiary to bvsup? (old vers. of suds?)
svco        :Co-routine.
svd         :Deprecated routine; EISPACK version instead of LINPACK.

```

Goal Files. Goal files are basically a concatenation of all of the out7a files from SHOW7 (as modified for exception testing). For the inclusion testing to be declared a success, the goal file generated during the validation must agree identically with the one contained in `vallib4.a`. *Caution:* Note that the first line of a goal file must be blank.

Following are the goal files as of March 1996:

==> goalimsl <==

```
3: blib is disjoint from craylib
6: mdr equals doc within exception left and right
8: doc is contained in blib within exception left
```

==> goalmath <==

(MATHLIB)

```
3: blib is disjoint from craylib
5: adv is contained in mdr
6: mdr equals doc
8: doc is contained in blib
```

==> goalmssl <==

```
4: mdr is contained in sup
5: adv is contained in mdr
6: mdr equals doc
7: sup equals src
```

==> goalmssl3 <==

```
4: mdr is contained in sup
5: adv is contained in mdr
7: sup equals src
```

==> goalnag <==

```
3: blib is disjoint from craylib within exception
5: adv is contained in mdr
6: mdr equals doc within exception left and right
8: doc is contained in blib within exception left
```

==> goalpmath <==

```
1: sup is contained in blib7
3: blib is disjoint from craylib
4: mdr is contained in sup within exception left
5: adv is contained in mdr
6: mdr equals doc within exception left
7: sup equals src within exception left
8: doc is contained in blib
```

==> goalslatec <==

```
1: blib is contained in sup
2: sup-blib is contained in craylib within exception left
3: blib is disjoint from craylib within exception
4: mdr is contained in sup
5: adv is contained in mdr
6: mdr equals doc
7: sup equals src
8: neither doc nor blib is contained in the other
9: doc-blib is contained in sup-blib
```

---

⁷ Note that the Cray binary PMATH is special, in that for efficiency reasons it contains CAL-coded random number generators, rather than the C versions in the source archive. This adds extra entry points.

4.4. Adding a New Library to the Validator

This section describes the steps required to add a new library to the NMG Validator.

4.4.1. Update Main Scripts

Suppose one wishes to add library ABC to the Validator. One must first update `liblist`, to add ABC to the menu, and then add an `if` test for ABC to `startval`.

Next modify `lstmake` by adding an `if` block for ABC. This block sets paths to library components of ABC and declares when components are empty. Also add an `if` statement to the bottom of `lstmake` to get word counts on library components.

Sometimes the script that builds a particular component must be modified for special purposes. One such modification can be seen in the binary component (`blib`) for library IMSL. Here the problem was that IMSL had three binary libraries, not just one.

4.4.2. Create a Goal File

Creating file `goalabc` is probably the trickiest part of adding ABC to the Validator. Initially ignore that `goalabc` has not been created. After one run of Validator, an approximate `goalabc` can be created by opening `v_report` with an editor and deleting all lines except the inclusion comparison lines. (These all begin with a number in the extreme left position followed by a colon.) Add one blank line at the beginning of this file, so that it looks like:

```
1:  something something
2:  something something
```

Copy this file into `goalabc` and store `goalabc` into `vallib4.a`.

The file `goalabc` must be iterated upon. For example, certain goals are probably not satisfied, and possibly exception files must be built. Another way of getting version one of `goalabc` is to copy the goal file from another library into `goalabc`, store in `vallib4.a`, run the Validator, and iterate.

4.4.3. Create Exception Files

Suppose the librarian decides that inclusion test 1 should come out as "component A equals component B", but when first validated this does not occur. Suppose that A-B is not empty (exception left). If the librarian wants to see the output of this test completely, he/she extracts file `diff1` from `dfiles.ABC`. In `diff1`, A-B is shown as well as B-A and the intersection. In working with certain libraries, errors have been discovered by examining these list differences. Possibly some or all of the names found in the list should not have been there to begin with. (For example, in a library update certain old routines were not eliminated, but they were supposed to be.)

If the librarian decides that all the names of A–B are exceptions which are legal, then these names along with annotation become `exc11`.¹ The names in A–B can be found in a vertical list found in `sfiles.ABC` in `vfiles.ABC`. The name of the file is `sym1-d1`.

In general, exception files have names of the form `exc1n`, where *l* is the test number and *n* has one of the following values. The associated `sfiles`-file is `sym1-dn`.

<code>n = 1</code>	left exception
<code>n = 2</code>	middle or intersection exception
<code>n = 3</code>	right exception

If validator is run on ABC without an orphan list and orphans exist, an error will be reported, and the orphan list will appear in the orphan report of `v_report`. This is usually an indication of an error. If the librarian decides that the orphan list is legitimate, it is copied to file `orphlist` and annotated by the librarian. The only example of an orphan list at present is stored in `efiles.SLATEC`.

If needed, the file `efiles.ABC` should contain a README file, `READ.exception`, `exc-files` and/or `orphlist`. *Caution:* If one constructs an `efiles.ABC`, files must be extracted from it in `lstmake`. (See the SLATEC block, for example.)

---

¹ See Section 4.3.3 for an example of the suggested format for annotations.

4.5. Back-up Procedures

This section describes the back-up directory and file structures for the NMG Validator.

4.5.1. ar-file Contents

The previous sections have mentioned the two ar-files that are used to store files related to the Validator. Following are the contents of the current versions:

:::::::::::::	:::::::::::::
val.a	vallib4.a
:::::::::::::	:::::::::::::
showt4b.f	dupper
showta.f	list13
showt1a.f	goval
showt2b.f	mclean
showt7c.f	testsup
dupnamb.f	.lstmake
showt5.f	orphgen
README	goalpmath
	goalslatec
	startval4
	goalmsl
	goalmsl3
	goalnag
	goalimsl
	goalmath
	exceptest
	efiles.IMSL
	efiles.SLATEC
	efiles.PMATH
	READ.details
	efiles.NAG

Note that these need to be updated and a new `valyymmdd` file created and archived (see next subsection) whenever any of the scripts or Fortran codes is updated or whenever any goal or exception file is changed.

4.5.2. SCF Archive Directory Structure

NMG Validator back-up files are stored in shared directory `.m29/.nmg/validate` on the SCF archive. Its structure is as follows:

`/document` — copies of Validator documentation, in the form of ar-files containing examples and annotation.

`/results` — results of complete validations. This has subdirectories for results from different platforms. ar-file `resultsyymmdd`, contains the `v_report` and `vfiles.lib` files generated during the validation that was run on `yy-mm-dd`. (See Section 4.2.5.)

`/Cray-YMP` — validations run on the Y-machine;

`/Cray-J90` — validations run on the J90 cluster.

`/runit` — copies of the files needed to run a validation. This too has subdirectories for different platforms, because some goal or exception files may be platform-dependent. Some scripts may also have platform-dependent differences. These have not yet been instrumented to use `CHANGE` (see Section 2.7).

The version of Validator updated on *yy-mm-dd* is stored in the appropriate subdirectory as ar-file *valyymmdd*. This contains `liblist`, `start-val`, `val.a`, `vallib4.a`, and a `README` file.

`/Cray-YMP` — Validator for the Y-machine;

`/Cray-J90` — Validator for the J90 cluster.

It is good practice to save the results of a complete passed validation periodically in the appropriate subdirectory of `.m29/.nmg/validate/results`, so that the details are available for comparison purposes if a future validation fails.

4.6. Notes on Future Development

This final section contains notes on desirable future development of the NMG Validator. Refer to Section 3.10 for related information.

- The current version of `lstmake` generates `craylib` whenever `blib` is nonempty. It might be more efficient to create it once and for all at the beginning.
- The orphan report, `orphan_out`, is saved too many times. It should be omitted from `lfiles.lib` (see Section 4.2.5).
- The NMG Validator currently only runs on the LC Cray's. Libraries should be validated on any platform serviced by MSS.
- Validator should be extended to include all checks noted in Section 4.1.
- We need better tools for diagnosing internal consistency of databases. (See T17A in Section 3.3.6 for a starting point.)
- We need a tool to validate the consistency of an edited `llast1` file. (See Section 3.2.1.) [Is this file really in the correct format? Are the labels properly nested, etc.?)