NMG Documentation

Part II. Programmer's Guide

Frederick N. Fritsch
Robert P. Dickinson, Jr.

RECEIVED
AUG 16 1996
OSTI

July 1996

db
DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

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.

DISCLAIMER

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

NMG Documentation

Part II. Programmer's Guide

Frederick N. Fritsch
Robert P. Dickinson, Jr.

Mathematical Software Service
Livermore Computing Department
Lawrence Livermore National Laboratory

This is the second 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.

Chapter 2. How It Works

2.0. Overview

NMG is an interactive utility to assist in the selection and fetching of mathematical software from the Fortran libraries maintained by the LC Mathematical Software Service (MSS). The NMG system consists of two C shell scripts and three Fortran 77 programs. Following is an alphabetical list of commands, with an indication of where they are processed.

advice — Advisor Program (Fortran)
end — Main Control Script (C shell)
fetch — Fetch Script (C shell)
help — Help Processor (Fortran)
mail — Main Control Script (C shell)
menu — MenuDriver Program (Fortran)
news — Main Control Script (C shell)
quit — Main Control Script (C shell)

Figure 1 shows the connectivity among these parts of NMG and the various data files they read from. More details on each component of NMG will be given in the relevant sections. NMGLIB is a collection of Fortran subprograms that are shared by the Help Processor, Advisor, and MenuDriver. It is described in a separate section before the programs that use it.

Contents

2.0. Overview	1
2.1. The Main Control Script	4
2.1.1. Usage Statistics.....	4
2.1.2. Locally Processed Commands.....	4
2.1.3. Directory Structure	5
2.1.4. Platform Dependencies	5
2.1.5. The nmg Program	6
2.2. NMGLIB	7
2.2.1. Routine Descriptions.....	7
2.2.2. Program Structure.....	8
2.2.3. Creating a Source Code	10

Figure 1. NMG Components and Data Files

2.3. The Help Processor	12
2.3.0. Overview	12
2.3.1. How It Works	13
2.3.2. File Formats	14
2.3.3. Platform Dependencies	16
2.4. The Advisor Program	17
2.4.1. Database Format.....	17
2.4.2. An Example	19
2.4.3. Contents of COMMON	21
2.4.4. How It Works	23
2.4.5. Platform Dependencies	24
2.5. The MenuDriver Program	25
2.5.1. Database Format.....	25
2.5.2. An Example	27
2.5.3. Contents of COMMON	31
2.5.4. How It Works	31
2.5.5. Platform Dependencies	32
2.6. The Fetch Script	33
2.6.0. Overview	33
2.6.1. Help Facility	34
2.6.2. Initialization Phase	34
2.6.3. Documentation Fetching.....	34
2.6.4. Source Code Fetching	34
2.6.5. Platform Dependencies	36
2.6.6. Exit Codes	36
2.6.7. Usage Statistics.....	37
2.7. The CHANGE Program	38
2.7.1. Basic Information	38
2.7.2. Instrumenting a Program for CHANGE.....	38
2.7.3. An Example	39
2.8. How to Install NMG on a New Platform	41
2.8.0. Basic Information	41
2.8.1. Installation Overview	41
2.8.2. Processing Scripts and NMGLIB	42
2.8.3. Installing the Help Processor	42
2.8.4. Installing the Advisor Program	43
2.8.5. Installing the MenuDriver Program	43
2.8.6. Preparing the Fetch Archives and Legal Notices	44
2.8.7. Validation and Clean-up	45
2.8.8. Tool Descriptions	46
2.9. Notes on Future Development	48

2.1. The Main Control Script

The main point of interaction between the user and NMG is a C shell script referred to as the Main Control Script.¹ This script is actually called `newnmg`, for historical reasons. It directly processes the `news`, `mail`, and `end (quit)` commands. `newnmg` passes control to another script or program for processing the other commands. It also generates NMG usage statistics.

`newnmg` may be initiated with zero or one argument. After a greeting and some initialization coding, it types a brief summary of available commands if initiated without arguments, then drops into the standard (infinite) `while` loop to process commands. A response containing a question mark to the standard "Command?" prompt breaks out of the loop to give the above-mentioned command summary. Other commands are processed via a `switch` statement. These are made case-insensitive by converting to lower case before the `switch`. With the exception of `mail` and `menu`, only the first letter of a command is examined.

2.1.1. Usage Statistics

As part of the initialization coding, `newnmg` writes the date, host name and user name to a temporary file (script variable `stat`). At each branch of the `switch` (except `default`), `newnmg` writes a single word (ALL CAPS) to the statistics file to indicate that the user has exercised that command. In the case of `fetch`, the name of the statistics file is passed to the Fetch Script so that it can append additional information. Upon execution of an `end (quit)` command, `newnmg` concatenates a copy of the mail file (if one exists) and mails the statistics file to "user m29".² (Note that this means that if a user kills `newnmg`, rather than terminating normally, the entire statistics record is lost.)

2.1.2. Locally Processed Commands

The `news` command is implemented by merely starting up UNIX utility `more` on file `nmgnews`. (The script doesn't even check for the existence of this file.)

The `mail` command is also implemented simply. At the first occurrence, the user name and date are written to a second temporary file (script variable `msg`). Whatever the user types is then appended to this file via the UNIX `cat` command. (The user is required to type control-D to terminate input.) If another `mail` command is given, the user's input is appended to the existing file. As noted above, the mail file is appended to the statistics file

¹ Refer to the subsection below titled *The nmg Program* for a caveat here.

² `m29` is an actual user name on the LC Cray Y-MP's, which is used solely for the purposes of collecting NMG usage statistics. This dates back to the old NLTSS system, in which fictitious user 000029 had been created for math library activities. On other systems, `m29` is set to the name of some member of MSS.

upon execution of an **end (quit)** command. At this point, the message is also mailed separately to "user HotLine", which is typically set to the LC Client Services HotLine plus some member of MSS. (As above, this means that if a user kills newnmng, rather than terminating normally, the entire mail message is lost.)

2.1.3. Directory Structure

The following directory structure is used by NMG. `$prefix` is the value of a newnmng script variable containing the path to the NMG directory structure. On most platforms, this is `/usr/local/math/nmg`.

Refer to the sections on the programs that use them for more information on the contents of these directories.

2.1.4. Platform Dependencies

The script itself is pretty much standard UNIX. There are four key script variables whose values may be platform-dependent:

- `prefix` the path to the NMG directory structure.
- `mail` the path to the mailer to use for **mail** and statistics.
- `m29` the place to send usage statistics.
- `HotLine` the place(s) to send **mail** messages.

newnmng is instrumented to use Joe Grcar's CHANGE program to control such platform-dependent details.³

³ These are documented in the newnmng script itself. Refer to Section 2.7 for more information about the use of CHANGE.

2.1.5. The `nmg` Program

To make sure that the NMG directory contents are not compromised by unauthorized accesses, users do not execute `newnmg` directly. Instead, a simple C-coded control program is used to initiate `newnmg`.⁴ Compile `nmgdr.c` via "`cc -o nmg nmgdr.c`", and install the executable `nmg` in `/usr/local/bin` or other suitable public directory. The NMG directory structure (script variable `prefix`) should have a suitably restricted group (`mss` is used on LC platforms) and all access to "others" denied. The executable `nmg` should be set to run as a member of this group via "`chmod 2711 nmg`".

⁴ The path to the NMG directory structure is built into the source code `nmgdr.c`. (This will need to be modified by hand, if necessary, since `CHANGE` does not work for C source codes.)

2.2. NMGLIB

The NMG Help Processor, Advisor, and MenuDriver programs share a common set of routines, referred to as NMGLIB. These are contained in directory nmglib/nmgsubs of the NMG distribution tar-file.

2.2.1. Routine Descriptions

Following are very brief descriptions of the routines contained in nmgsubs.

- GENTREE reads the data structures which are needed to represent a data tree.
- HELPS2 is the primary interface routine for the NMG help package.
- HFDISP displays a helpfile.
- HMNGEN reads a helpfile list-file and generates the associated help menu.
- LASTNB finds the index of the last nonblank character in a string.
- LEVPRT determines arrays NLEV and LLEV for GENTREE.
- LEVS1 prints as many levels of information as possible at current node.
- LEVSA contains most of the logic for traversing the tree, prompting the user for information concerning the next task to be executed, displaying information, etc. (Advisor version).
- LEVSM contains most of the logic for traversing the tree, prompting the user for information concerning the next task to be executed, displaying information, etc. (MenuDriver version).
- LLENL calculates the length of a label.
- MAINA is the main code for Advisor. (Contains some platform-dependent coding.)
- MAINH is the main code for Help. (Contains some platform-dependent coding.)
- MAINM is the main code for MenuDriver. (Contains some platform-dependent coding.)
- MMENUD presents the user with the main menu of categories to select from, prompts the user for the selection, and opens the requested intree file.
- MPROM2 requests a response from the user while examining the main menu.
- PRTND prints the information at a node.
- PRTND1 partially adds to the number of levels of text that the user sees when viewing at a particular node.
- PRTNDI prints an information screen for the user.
- READS reads a line from the terminal and returns the blank-separated symbols found in the line.

SRCH3 searches MTABLE for a given GAMS root.

SRCHG determines what lines contain descriptions of subroutines with a given GAMS classification, displays menu of libraries containing routines in this category, and prompts user for selection.

SRCHR searches the label list for a label supplied by the user.

SROUT1 displays descriptions of all routines with a given GAMS classification.

TWODIG handles two-digit numbers in GAMS labels.

2.2.2. Program Structure

The current structure of these codes is as follows. Here NAME is a subroutine name, and indentation indicates routines called by the un-indented routine. MAIN? is a main code.

Help Processor:

```

MAINH
  LASTNB
  HELPS2
    HMNGEN
      LASTNB
    LASTNB
    HFDISP
      READS
    READS

```

Advisor:

```

MAINA
  LASTNB
  READS
  HELPS2
    (see Help for subsidiaries)
  MMENUD
    LASTNB
    MPROM2
      HELPS2
        (see Help for subsidiaries)
      READS
    GENTREE
    LEVPRT
  LEVSA
    LEVS1
      PRTND1
        PRTND
          SRCH3

```



```

                                TWODIG
                        PRTND
                        SRCH3
                        TWODIG
                READS
                HELPS2
                        (see Help for subsidiaries)
                SRCHR
                        LLENL
                PRTNDI
                        READS
    
```

MenuDriver:

```

    MAINM
        LASTNB
        READS
        HELPS2
                (see Help for subsidiaries)
        MMENUD
                (see Advisor for subsidiaries)
        LEVSM
                LEVS1
                        (see Advisor for subsidiaries)
                READS
                HELPS2
                        (see Help for subsidiaries)
                SRCHR
                        LLENL
                SRCH3
                SROUT1
                        SRCHG
                                PRTND
                                SRCH3
                                TWODIG
                                READS
                                HELPS2
                                        (see Help for subsidiaries)
                GENTREE
                        LEVPRT
                LLENL
                READS
                HELPS2
                        (see Help for subsidiaries)
    
```

PRTNDI
READS

2.2.3. Creating a Source Code

All NMGLIB routines are written in standard Fortran 77. With the exception of LEVSA and LEVSM, the source code for NMGLIB routine *NAME* is in file *name.f* in subdirectory nmgsb. A C shell script *assmb1* is used to assemble the requisite routines from NMGLIB into a complete source code for one of the NMG Fortran programs.

Creating a Complete Source File Set. Two routines with similar functions (LEVSA and LEVSM) are actually contained in nmgsb file *levs.f*. This is instrumented for Joe Grcar's CHANGE program,¹ which must be run twice to produce the required separate source files *levsa.f* (Advisor blocks active) and *levsm.f* (MenuDriver blocks active) *before* running *assmb1*. (Otherwise, *assmb1* will fail due to a missing file when attempting to create an Advisor or MenuDriver source code.)

Aside: It was intended that LEVSA and LEVSM eventually be merged, with the code-dependent differences controlled by variable CODENM, as in MMENUD. This has not been done because of the drawback that Advisor would be larger than necessary, due to the necessity to load MenuDriver-specific routines that will never be referenced.

Perhaps a different alternative would be to have a single code that figures out which program is to be run by the format of the databases or via a signal when it is started up by the main control script.

Assembling and Compiling a Program. The *assmb1* script is used as follows:

```
assmb1 list source dir
```

where

list is a file containing a list of files to be merged into the source file;

source is the desired name for the source file;

dir is the name of the directory containing the files to be assembled.

One then compiles the source file as follows:²

```
f77 -o exec source
```

The standard values for italicized file-names given above are given in the following table. These list-files are contained in directory nmglb of the NMG distribution tar-file, as is subdirectory nmgsb. Utility *assmb1* and others are in directory nmglb/tools.

¹ This is documented in the source code. Refer to Section 2.7 for more information about the use of CHANGE.

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*.

Code	<i>list</i>	<i>source</i>	<i>dir</i>	<i>exec</i>
Help Processor	hlpsrclist	help.f	nmgsubs	hlpc
Advisor	advsrclist	adrive.f	nmgsubs	adrv
MenuDriver	mdrsrclist	mdrive.f	nmgsubs	mdrv

SLATEC Compliance. The fact that change blocks are delimited by lines that begin 'C*****' means that LEVSA and LEVSM are not truly SLATEC format, which requires that only SLATEC lines begin 'C***'. This remark also applies to the main*.f files, discussed in subsequent sections. (This minor annoyance could be removed by a modified version of CHANGE which eliminated inactive blocks and CHANGE delimiters from the output source code.)

2.3. The Help Processor

To avoid redundancy and reduce maintenance difficulties, the three Fortran programs that support NMG share a common set of helpfiles and a set of routines that read and display them. Each program has an associated list-file that indicates which helpfiles are accessible from this code.

2.3.0. Overview

Subdirectory `help/` of `NMGDIR`, the path to the NMG directory structure, contains the NMG Help Processor and the necessary files and directories to support this function. These are:

<code>hlpc</code>	the executable code that implements the NMG <code>help</code> command.
<code>advlist</code>	the list of helpfiles available from Advisor.
<code>mdrlist</code>	the list of helpfiles available from MenuDriver.
<code>nmglst</code>	the list of helpfiles available via the NMG <code>help</code> command.
<code>helpfiles/</code>	the helpfiles themselves.

To illustrate helpfile sharing, following are listings of the `helpfiles` directory and the three list-files as they currently exist:

```
milstein% ls helpfiles
advmen advstd helpov libava mdrcat mdrngen mdrlst nmgov
advov  fetch  libabb mailov mdrdsc mdrlib mdrov

milstein% more *list
:::::::::::::
advlist
:::::::::::::
advhelp
Advisor Help Screens -- Revised 5 February 1993 (F.N.Fritsch)
advov:  Advisor Overview
advmen: Main Menu Commands
advstd: Standard Screen Commands
libabb: Library Abbreviations
libava: Library Availability
helpov: Help Overview
:::::::::::::
mdrlist
:::::::::::::
mdrhel
MenuDriver Help Screens -- Revised 13 May 1994 (F.N.Fritsch)
mdrov:  Opening screen
mdrgen: General Information
mdrcat: Category Search Commands
mdrlib: Library Selection Display
mdrlst: List Continuation Commands
```

```

mdrdsc:  Description Format Notes
libabb:  Library Abbreviations
libava:  Library Availability
helpov:  Help Overview
::::::::::::
nmglist
::::::::::::
nmghelp
NMG Help Screens -- Revised 13 May 1994 (F.N.Fritsch)
nmgov:   NMG Overview
helpov:  Help Overview
advov:   Advisor Overview
mdrov:   Menu Overview
fetch:   Fetch Overview
mailov:  Mail Overview
libabb:  Library Abbreviations
libava:  Library Availability

```

Note that mdrov appears in both mdrlist and nmglist, but with different titles. Note also that libabb, libava, and helpov appear in all three lists, but not necessarily in the same order.

2.3.1. How It Works

Subroutine HELPS2 is the main interface to the NMG Help Package. It is called by MAINH, the Help Processor main program, and by various Advisor and MenuDriver subroutines. Its calling sequence is:

```

CHARACTER*6  NAMSF1
CHARACTER*56 HFLST, HFDIR
CALL HELPS2 (NAMSF1, HFLST, HFDIR)

```

where

```

NAMSF1  is the name of the helpfile containing the first help screen to be presented,
        assumed to be in directory HFDIR;
HFLST   is the path to the helpfile list-file (i.e., to the appropriate one of the list-files
        mentioned above);
HFDIR   is the path to the helpfiles directory.

```

The basic function of HELPS2 is to display the helpfile indicated in NAMSF1 and then ask the user whether he/she would like to view any of the other available helpfiles.

The first time HELPS2 is called in a given application, it calls HMNGEN, which uses HFLST and HFDIR to construct file HFDIR/helpmenu that will be displayed as the "help menu" for this application. Note that the order of names in the list-file determines the order in which they will appear in the menu. As noted above, the same helpfile may have different menu titles in different lists, if desired. Following is the helpmenu file generated from nmglist (i.e., the one used by the NMG help command).

Generated help menu

NMG Help Screens

1	(nmgov)	NMG Overview
2	(helpov)	Help Overview
3	(advov)	Advisor Overview
4	(mdrov)	Menu Overview
5	(fetch)	Fetch Overview
6	(mailov)	Mail Overview
7	(libabb)	Library Abbreviations
8	(libava)	Library Availability

(Refer to the **help** example, Section 1.3.2 for the prompt that is displayed when this help screen is presented to the user.)

The file named in NAMSF1 is then displayed via a call to HFDISP, after which HELPS2 issues the prompt

Help menu / Return / End :

The case of the response is ignored. If the first letter is "e" or "q", HELPS2 executes a STOP statement. If it is a carriage return or begins with "r", HELPS2 returns. If the response begins with "h", HELPS2 calls HFDISP to display the help menu generated earlier by HMNGEN. The user is then asked to type the number or name of the desired help screen. The code then cycles back to the standard prompt display after another call to HFDISP.

As described in detail below, the helpfiles are simply text files. The only tricky thing in HFDISP is that a line containing "p/" in the first two characters is interpreted as a page break sentinel. The routine pauses for the user to read the screen as displayed so far and expects the user to type "more" to display more of the text. (This is a concession to old 24-line terminals which is probably no longer needed.)

2.3.2. File Formats

The helpfile list-file, HFLST, is assumed to have the following format:

Line 1: Ignored by help code. (An earlier version of the Help Package used this as the name for a larger file containing the merged helpfiles.)

Line 2: Title line for the help menu. (Characters after "--" have revision information and are ignored by HMNGEN.)

Subsequent lines contain two fields, separated by a colon (:).

The first field is the name of a helpfile that can be accessed by this program; the second field is identifying text to be displayed for this screen in the help menu (copied verbatim, including any whitespace).

Helpfile format:

Line 1: Ignored by HFDISP. (This was a title line for the old Help Package, but it has been superseded by the title contained in the associated list-file.)

It is recommended that this line contain a brief description and a revision date, with the initials of the person who updated it last.

Subsequent lines contain the help screen text, which is displayed verbatim.

The file may terminate with the following line (start in column 1):

```
**** End Help File ****
```

If present, any following lines are ignored and may contain internal documentation or other notes.

As an example, the following is a listing of helpfile nmgov, the first one displayed by the NMG **help** command. (Refer to the **help** example in Chapter 1 to see how this is presented to the user.)

NMG Overview

Revised: 95-12-28 (FNF).

The Numerical Mathematics Guide, nmg, provides access to the mathematical libraries available on the LC Cray's. The commands available are:

```
advice - offers advice on routine selection.

menu    - provides access to a complete menu of all user-level routines
          in the LC mathematical libraries.

fetch   - provides access to online documentation and/or source code
          (when available) for individual routines.

mail     - provides a mechanism for sending a message to the maintainers
          of NMG.

help     - provides access to the NMG help facility.

news     - provides news on recent changes to NMG.

end      - terminates NMG. ("q" or "quit" is a synonym.)
```

This prompt means there is more to this help screen. Type "m" to see it.
p/

Commands may be typed in upper or lower case. Only the first one or two characters (enough to specify a command unambiguously) need be typed. After completion of any command except "end" or "quit", NMG will loop back to the standard "Command?" prompt for a new command. (Type "?" for a reminder of the available commands.)

NMG has extensive help facilities. The use of this utility is designed to be self-explanatory. The Advisor, MenuDriver and Help processor have a common prompt convention: The prompt is a list of the acceptable commands, with the key letter (the first) capitalized. Type the key letter to select a command. Throughout NMG, "q" is equivalent to "e" or "end".

The first NMG command may be on the command line. Thus,
nmg help
will go directly to this help facility.

The prompt below means that you should type "h" for a menu of additional help screens.

**** End Help File ****

2.3.3. Platform Dependencies

The program itself is standard Fortran 77. There are three key variables whose values may be platform-dependent:

HLPDIR	the path to the NMG help directory structure, namely NMGDIR/help.
HFLST	the path to the nmglst file.
HFDIR	the path to the helpfiles directory, which contains the helpfiles.

MAINH, the Help Processor main program, is instrumented to use Joe Grcar's CHANGE program to control HLPDIR.¹ The other two variables are defined in terms of HLPDIR.

¹ This is documented in the source code. Refer to Section 2.7 for more information about the use of CHANGE.

2.4. The Advisor Program

The NMG Advisor is a portable Fortran 77 program that displays advice on the routines in the libraries supported by MSS via a modified decision tree structure. Unlike its predecessor, LIP,¹ the Advisor tries to display as much of the decision tree as will fit on a reasonably-sized screen. (At present, the maximum number of lines is set to 22, as a concession to old 24-line terminals, but this is a parameter that can be easily modified. Ideally, this parameter ought to adapt itself to the user's environment, but such a facility has not been implemented.)

Advisor starts by displaying a main menu containing a relatively small number of main categories. On user command, it then opens the appropriate database and displays the tree to whatever depth will fit in the allowed screen size. The basic rule is that a level will not be expanded unless all its branches will fit on the screen. The user is then given the option to display a particular branch in more detail, back up to the previous screen, or return to the main menu. The terminal nodes (leaves of the tree) are assumed to contain specific advice, in the form "Use *ROUTINE (LIBRARY)*".

Any node may also be an "information node", which means that it contains additional information that might aid the user in interpreting the choices available at this branch of the tree.

2.4.1. Database Format

The Advisor main menu is kept in file `menua`, which is a straight text file except for the first line, which contains the number of lines of text to be read (in I3 format).

Advisor uses a separate database for each main menu subject. (There are eight right now, numbered 1–8.) For example, the database (or decision tree) for subject 4 (Interpolation/Approximation) is in file `intree4`.

The database for each subject has the following format. (The names are the variables into which this information is read by subroutine `GENTREE`.) This same format is used for the first part of a `MenuDriver` database.

- The first line contains five numbers (5I6), followed by identification information:
 - ND number of nodes in the tree.
 - NC number of pointers from parent to children. Because this is a tree, not an arbitrary graph, $NC = ND + NT - 1$, where NT is the number of terminal nodes. [The term NT is there because terminal nodes are indicated by null pointers, which count as one child each. The "-1" is because nothing points to the root node.]
 - NL number of lines of text that contain the information about the nodes.

¹ LIP = Livermore Interactive Program. This was written by Tokihiko Suyehiro and was based on the work of Pat Gaffney, then at ORNL.

INLEV not currently used by Advisor or MenuDriver.

NLMD number of nodes that are information nodes.

Convention: Columns 36–80 of this first line, while ignored by GENTREE, should contain information to identify the update status of this database. The following format is used by the database update tools (see Chapter 3):

col. 36–41: `adrive` if this is an Advisor database, or

`mdrive` if this is a MenuDriver database.

col. 43–50: date this database was last changed (in `yy/mm/dd` format).

col. 52–54: initials of the person who did this last update.

- The next set of lines contains IC, the complete list of children (10I6). IC contains NC entries. It consists of ND sublists, each of which contains the node numbers of the children of the current node. A node whose IC list is null (consists only of zero) is a terminal node.
- Next comes ICP, a list of pointers into IC (10I6). It contains ND entries. ICP(I) is the index of the first child of the I-th node, and ICP(I+1)–1 is the index of the last child of the I-th node, where we use the convention ICP(ND+1)=NC+1. A terminal node has ICP(I)=ICP(I+1)–1 and IC(ICP(I))=0.
- Next comes IPP, a list of pointers into the text lines that follow (10I6). It contains ND entries. If we assume the text lines are numbered from 1, IPP(I) is the index of the first line of the text for the I-th node, and IPP(I+1)–1 is the index of the last line, where we use the convention IPP(ND+1)=NL+1. **[Caution:** This assumes that nodes are numbered consecutively from 1 and that the text is given in node number order. The present Advisor assumes further that the nodes are numbered width-first. That is, the children of node 1 (the root) are nodes 2,3,...; next come the children of node 2; etc.] This data structure allows an arbitrary amount of text (subject to screen size) at each node.
- Next comes IP, the NL lines of text that contain information about the nodes. These are simply the lines of text that are to be printed at associated nodes of the tree. These contain information to enable the user to narrow down the problem domain. Except for information block text, which is displayed separately, the printable field is limited to 64 characters per line, to allow space for printing node labels and indenting to show sub-levels.

Convention: Terminal nodes in an Advisor database contain advice on specific routine(s) to use. Some validation tools (see Chapter 3) assume that this advice is in one of the following formats, where ^ denotes one or two leading spaces:

^Use ROUTINE (LIBRARY) .

^Use ROUTINE (LIBRARY) or

^Use ROUTINE1 and ROUTINE2 (LIBRARY) .

^Use ROUTINE1, ROUTINE2, and ROUTINE3 (LIBRARY) .

The first form is the most common (a single routine from a single library). The parentheses around the library name are required. The final period is optional, but recommen-

ded. Routines from different libraries must appear on separate lines, and the word "Use" must be capitalized in each.

Convention: In order to make the text portion of a database more human-readable, the portion of these lines starting in column 71 is reserved for identifying information, which is written there by the database update tools (see Chapter 3). Line IPP(I) contains the label for node I, starting in column 71. Subsequent printable lines have this field blank. Information block lines have "i" in column 71. This means that this information is limited to 70 characters per line. [These are not read by any of the routines that process and display databases.]

- If NLMD (number of line-count modifications) is nonzero, there follow NLMD pairs of numbers LMD1, LMD2 (2I6) which define the information blocks in a rather negative manner. For each J, LMD2(J) is the number of print lines for node number I=LMD1(J). This means that only lines IPP(I) through IPP(I)+LMD2(J)-1 are to be printed when displaying node I with an "expand" command. Lines IPP(I)+LMD2(J) through IPP(I+1)-1 thus constitute an "information block" for node I that is accessed via the "information" command. Note: Must have LMD2(J) < IPP(I+1)-IPP(I).
- Finally, there appears a table of node numbers and associated labels (I4,2X,A24):
 NDDUM node number. (While these are generally consecutive integers, they need not be. In fact, this field is not even read by GENTREE.) **Caution:** The index I in the descriptions of ICP and IPP actually refers to the position in this list, not the node number.
 LB node label. These consist of alternating letters and numbers.² The first character of the labels in *intreen* is *n*. The labels for the children of a node have an additional letter or number added. For example, the children of the parent node labeled 1 may have labels 1a, 1b, etc.; the children of 1a would be 1a1, 1a2, etc. (For Advisor databases, these were originally constructed by the appropriate Advisor tool from the connectivity information.)

2.4.2. An Example

Following is a simple Advisor database:

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			
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

² Because some GAMS categories contain two-digit numbers, the numbers in node labels may be one or two digits.

[f1(x)]**2 + [f2(x)]**2 + ...	i
Consult the interpolation/approximation tree for such routines.	i
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 the root is known	3a2b
Use C05AJF (NAG)	

3	2
1	3
2	3a
3	3b
4	3a1
5	3a2
6	3a2a
7	3a2b

The first three lines and the node label table at the end define the following tree, where we have placed the node numbers below the boxes and the node labels inside them. Because the corresponding elements of IC are zero, nodes 3, 4, 6, 7 are terminal nodes (i.e., have no children).

The value of NLMD is 1, so there is a single pair of numbers after the text, which tell us that node 3 (with label 3b) is an information node which has only two lines that are to be displayed normally. Here is what Advisor displays when this tree is expanded from the main menu. Note that although the tree information appears in the database in width-first order, it is displayed in depth-first order.

```

x  3      Solutions of Nonlinear Equations
x  a      Single equation in one unknown
      a1   All roots of a polynomial equation
           Use RPZERO (SLATEC)

```

```

x      a2      General equation
      a2a      An interval in which the desired root lies is known
                Use ZEROIN (MATHLIB) or
                Use SZERO  (PMATH).
      a2b      Although no such interval is known, some estimate of
                the root is known
                Use C05AJF (NAG)
i      b      System of equations
                Use SNSQE (SLATEC)
eXpand label / Info. label / Main / Previous / Help / End :

```

Nodes 3a and 3a2 are nonterminal nodes, so are marked "x" to indicate that they are expandable. If the user types "x a", the following display results. Note that the leading characters of the node label are suppressed to reduce the amount of typing required by the user to enter label information.

```

x      a      Single equation in one unknown
      1      All roots of a polynomial equation
                Use RPZERO (SLATEC)
x      2      General equation
      2a      An interval in which the desired root lies is known
                Use ZEROIN (MATHLIB) or
                Use SZERO  (PMATH).
      2b      Although no such interval is known, some estimate of
                the root is known
                Use C05AJF (NAG)
eXpand label / Info. label / Main / Previous / Help / End :

```

Because this is such a simple tree, it all fits on one screen, so no new information has been revealed by this expansion. Had any of the displayed nodes not been fully expanded in the earlier view, however, they would be further expanded in this view. Typing "p" returns us to the previous display.

Node 3b is marked "i" to indicate an information node. If the user types "i b", the following display results. Note that this includes *all* of the information at the node, not just the lines marked "i" in the database.

```

System of equations
Use SNSQE (SLATEC)
An alternate way of solving a system of the form
  f1(x) = 0, f2(x) = 0, ... (n equations in the n-vector x)
is to find a minimum of the sum of squares
  [f1(x)]**2 + [f2(x)]**2 + ... .
Consult the interpolation/approximation tree for such routines.
Return / End :

```

2.4.3. Contents of COMMON

The routines that need access to the data structure implied by the above-described database format contain the following declarations:

```

INTEGER ND, NC, NL, NLMD
INTEGER IC(1000), ICP(1000), IPP(1000)
INTEGER LMD1(500), LMD2(500), LMOD(500)
INTEGER NLEV(500), LLEV(500)
COMMON ND, NC, NL, NLMD, IC, ICP, IPP, LMD1, LMD2, LMOD, NLEV, LLEV
C
CHARACTER*100 IP(1200)
CHARACTER*24 LB(500)
CHARACTER*8 CODENM
COMMON /CHRCR/ CODENM, LB, IP

```

Variable CODENM is assumed to contain either "Advisor" or "Mdriver", to indicate whether the code being run is the Advisor or MenuDriver. This is set by the main code.

The remaining information is set up by GENTREE. In addition to variables read from the database, as described above, GENTREE computes the following (all of length ND):

- LMOD is initialized to zero, and then NLMD of the entries are changed according to $LMOD(LMD1(J)) = LMD2(J)$. (Note that array LMD2 is really not needed after LMOD has been set.)
- NLEV number of levels from this node that can fit on one screen.³
- LLEV number of lines that will be printed if the tree is expanded NLEV levels at this node.⁴

It should be noted that the dimensions of the arrays are somewhat arbitrary and are not internally consistent.

- Arrays ICP, LMOD, NLEV, LLEV, and LB should all have the same dimension $\geq ND$; array IPP should have dimension $\geq ND+1$, since $IPP(ND+1)$ is set to $NL+1$ in GENTREE.
- Array IC should be dimensioned $\geq NC$. This is somewhat larger than ND, since $NC = ND+NT-1$, where NT is the number of terminal nodes, $1 \leq NT < ND$.
- Array IP should be dimensioned $\geq NL$. This should be larger than the first set of arrays, since the ratio of NL to ND will be the average number of lines of text per node.
- The character length for IP has been set at 100 to allow for long labels. Since the labels start in column 71 and their length is limited to 24 characters by the declaration for LB, 94 would have actually been sufficient. (96 would probably be even better, since it is a multiple of eight.)
- Arrays LMD1 and LMD2 should both have the same dimension $\geq NLMD$, the number of information nodes. Although $0 \leq NLMD \leq ND$, NLMD is generally much smaller than ND. Furthermore, there clearly cannot be 500 information blocks in 1200 lines of text!

³ This is currently limited to 4 (although parts of the code will allow NLEV=5.)

⁴ This is determined by variable MAXLIN=21 in LEVPRT.

2.4.4. How It Works

Help processing. Help is available at various points in Advisor. In each case, HELPS2 is called with its first argument equal to the name of the help screen that is most relevant at this point. See Section 2.3 for details on how HELPS2 works.

Prompts and response processing. Whenever Advisor expects a response from the user it first writes a prompt that contains the allowable responses at this point. (Exception: "quit" is synonymous with "end", but is never explicitly listed in a prompt.) In most cases a single letter is sufficient to distinguish among the allowable responses, and the letter that is expected is capitalized. Because a user is likely to type "expand" instead of "x", a special check of the second letter is included if the first letter is "e". The user's response is read and broken into blank-delimited symbols by subroutine READS.

The main code is basically an infinite loop consisting of a call to MMENUD followed by a call to LEVSA. The only way to terminate is for the user to respond "end" to a prompt in one of these routines. We take for granted that most prompts allow the response "end" or "help", and do not mention these choices further.

Main menu processing. The first screen displayed by Advisor is the main menu, which is processed by MMENUD in a relatively straightforward manner. It contains logic to only read file menua the first time it is called; subsequent invocations will re-display the main menu from the CHARACTER*80 array IM (which is limited to NLMAX=100 lines).⁵ It then calls MPROM2 to prompt the user for a letter to indicate which tree is to be expanded.⁶ Any database file which is open at this point will be closed; then the response is checked against a built-in list for validity. If valid, the new database file is opened, GENTREE is called to read the database and set up COMMON, and the routine returns to the main code.

Screen display. Subsequent screens are displayed by LEVSA. After some initialization, LEVSA displays the information from the top node of the tree via a call to LEVS1. It then allows the user to type one of the commands "x" (expand), "i" (information), "p" (previous) or "m" (return to the main menu).⁷ Whenever a command includes a label, SRCHR is called to determine whether the requested label (when catenated to the current root label) is contained in LB. If so, the node number (index in LB) is returned; otherwise, an error message is issued.

The "previous" command is implemented via a push-down stack that is maintained in LEVSA. This contains the arguments to LEVS1 and is initialized to the information for the root of the tree with index KPREV=1 on entry. KPREV is incremented at each "expand" command and decremented at each "previous" command. If KPREV becomes zero, LEVSA returns to the main code, which re-displays the main menu.

⁵ Since the main menu is expected to fit on a screen, this is clearly an over-generous limit.

⁶ For historical reasons, MPROM2 allows the user to type either "x" (expand) or "s" (show).

⁷ For historical reasons, LEVSA allows the user to type either "x" (expand) or "s" (show).

The only really complicated part of Advisor is the logic to determine which nodes will be displayed if the entire subtree from the given node will not fit on the screen.⁸ This is handled by LEVS1, which has arguments NO, the index of the node to be displayed, LEN0, the number of characters in its node label, and LBO, the node label itself. A maximum of 4 levels will be displayed, with the rule that a node will not be expanded if all of its children will not fit on the screen. Arrays NLEV and LLEV are used to determine which nodes will be displayed.

PRTND does the actual printing of the information at a node. It prints an "x" in column 1 of lines that are non-terminal ($IC(ICP(I)) \neq 0$) and an "i" in column 2 of information nodes. Node labels are indented two characters for each level below the top-level node. The root of the current subtree (argument LBO, above) is suppressed from displayed node labels and catenated onto the user's response to produce complete labels for SRCHR. PRTND1 adds additional levels of printing to the display.

The "information" command is handled by a call to PRTNDI, which basically prints the entire block of text associated with the requested node. The only complication is that the characters "p/" in columns 1-2 act as a page break, which causes a pause until the user reads the text and responds to a prompt. (These page breaks are added by hand by database maintainers to break up extremely long information blocks at logical points.) When PRTNDI returns, the current screen is re-displayed via a call to LEVS1.

2.4.5. Platform Dependencies

The program itself is standard Fortran 77. There are three key variables whose values may be platform-dependent:

NMGDIR	the path to the NMG directory structure.
HFLST	the path to the advlist file.
HFDIR	the path to the helpfiles directory, which contains the helpfiles.

MAINA, the Advisor main program, is instrumented to use Joe Grcar's CHANGE program to control NMGDIR.⁹ The other two variables are defined in terms of NMGDIR.

⁸ A screen is defined to be a fixed number of lines, which is built into LEVS1 as a parameter MAXP=22 (to allow two lines for prompt on an old 24-line terminal screen). Ideally, the code should be able to sense the user's environment and adjust MAXP accordingly.

⁹ This is documented in the source code. Refer to Section 2.7 for more information about the use of CHANGE.

2.5. The MenuDriver Program

The NMG MenuDriver is a portable Fortran 77 program that displays descriptions of all user-callable routines in the libraries supported by MSS. The program operates just like the Advisor initially, but the databases are organized according to the GAMS classification scheme.¹

Like Advisor, MenuDriver starts by displaying a main menu, which contains a subset of the GAMS main categories. On user command, it then opens the appropriate database and displays the tree to whatever depth will fit on the screen. The same commands that Advisor uses to traverse a tree are available in MenuDriver.

As with Advisor, any node may be an "information node", although very few of these exist in the current MenuDriver trees.

Nodes that contain routine descriptions are specially marked as "show nodes"; a "show" command will first display a small menu indicating which libraries have routines in this category. The user is then invited to select a library, or to display all available descriptions. As in the tree-display phase, only as many descriptions as will fit on a screen will be displayed at one time, and the user is expected to type "more" to continue the display. Descriptions are treated as units, and will not be split between screens.

2.5.1. Database Format

The MenuDriver main menu is kept in file `menum`, which is a straight text file except for the first line, which contains the number of lines of text to be read and a list of the names of libraries that are available on this platform (I3,5X,9A8).

MenuDriver uses a separate database for each main GAMS classification scheme category in which one of our libraries has routines. (There are fourteen right now, with letters A, C-L, N, R, Z.) For example, the database for category E (Interpolation) is in file `intreeE`. (Note the capital letter!)

The database for each category consists of two independent, but related parts. The first part describes the GAMS classification scheme (down to the level of detail that we have decided to include). Since this is in basically the same format as the Advisor decision trees, it will not be described in detail here. (See Section 2.4.) The only difference is that the first line contains the following additional information:

col. 56-61: MTABLE

col. 64-71: date the second (MTABLE) part of the database was last updated (in yy/mm/dd format). This should agree with the information in the first

¹ GAMS = Guide to Available Mathematical Software. The GAMS classification scheme was developed at the National Institute for Standards and Technology and has become a standard for classification of mathematical software.

line of the second part, but is repeated here so it is easy to find the complete version information by simply looking at the first line of the file. Note that, except for the MTABLE date, this first part of a MenuDriver database is rarely modified.

The second part, referred to as the MTABLE part, contains the actual short descriptions that are the real contents of the menu for this category. It consists of a table of text pointers, which is read by GENTREE, followed by the actual routine descriptions, read by SROUT1.

- The first item is NLP, the number of entries in the following line-pointer table (I6).

Convention: Columns 10–26 of this first line, while ignored by GENTREE, should contain information to identify the update status of the MTABLE part of this database. The following format is used by the database update tools (see Chapter 3):

col. 10–16: MTABLE:

col. 19–26: date the MTABLE part of the database was last updated (in yy/mm/dd format).

- Next comes a table of pointers into the text lines that follow (I6,4X,A8,2X,I6,2X,A8):
 LPTXT line number of the first line of text for the descriptions of routines in this category and library. These must be increasing, corresponding to the order of the text that follows. Counting starts with the first line of the text block as line 1.
 GCODE associated node label (GAMS category code). This must be one of the LB's listed in the first part of the database. These labels are assumed to be in lexicographic order.
 NRTNS number of routines in this category and library.
 LIBNM name of the library. For each category, these names are assumed to be in lexicographic order.

There is usually more than one entry for a given GAMS category, but each (GCODE, LIBNM) pair is unique. The last line of this table has GCODE=' END' (right-justified) and LPTXT equal to one greater than the total number of lines of text that follow. [Note that LPTXT(I+1)–LPTXT(I) is the number of lines of text for (GCODE(I),LIBNM(I)).]

Note: This table is the only connection between the two parts of the database, and we may decide at a later date to make them two separate files to simplify maintenance. Note that only the node labels from the first part appear here.

- The actual text that makes up the library menu for this main category follows. This text is broken into blocks. Each block consists of a collection of descriptions of routines that have the same GAMS classification (to the level supported by this database) and are in the same library. Each block begins with a header line of the form:

```

      libnm          GAMS :      gcode
where libnm (library name) is in columns 11 through 18;
      GAMS :   is in columns 31 through 35;
      gcode   (classification label) is in columns 41 through 48.
```

Each header is followed by a blank line. Each routine description is followed by a blank line. Routine descriptions are given in a format that was adopted from that used in the SLATEC 4.1 table of contents file. These are divided into two fields.

At the left (columns 1–10) appears a list of equivalent routine names, each in the format *NAME-X*. *NAME* is restricted to at most eight characters, and *X* is one of the following:

A = ALL (one routine for all types, or typeless).

S = SINGLE PRECISION;

D = DOUBLE PRECISION;

8 = REAL*8 (PMATH only);

C = COMPLEX;

I = INTEGER;

H = CHARACTER;

L = LOGICAL;

P = Package name.

The right-hand field (columns 12–80) contains a short description (generally limited to a maximum of six lines).

2.5.2. An Example

Following is a simple MenuDriver database. Even though this is the simplest database we have that illustrates the points we wish to make, it still extends over several pages. We have included line numbers to help with the correspondence between the MTABLE information and the following text.

An examination of the first part of the database (lines 1–14) illustrates the fact that, although node labels must be in increasing order, not all possible children need be present. This tree contains no nodes with labels A1, A2, or A5. (The official GAMS classification scheme has these categories, but they have been omitted because there are no routines in these categories in any of our supported libraries. Also, because of the small number of routines, we have not included the subcategories of A3, A4, or A6.)

The last line of MTABLE is line 23, so one must add 23 to the LPTXT values to get the line numbers of the associated headers. Note that the blank last line is necessary to properly terminate the last description.

1	5	8	5	0	0	mdrive 10/07/93 FNF MTABLE	02/13/96
2	2	3	4	5	0	0	0
3	1	5	6	7	8		
4	1	2	3	4	5		
5	Arithmetic						A
6	Real						A3
7	Complex						A4
8	Change of representation (incl. prime factorization)						A6
9	Sequences (e.g., convergence acceleration)						A7
10	1	A					
11	2	A3					

```

12      3  A4
13      4  A6
14      5  A7
15      8  MTABLE: 02/13/96
16      1  A3      1  NAG
17      6  A3      6  SLATEC
18     26  A3      2  mssl3
19     33  A4      7  NAG
20     56  A4      1  SLATEC
21     60  A6      2  SLATEC
22     68  A7      5  NAG
23     85      END
24      NAG      GAMS:  A3
25
26 F06BLE-S      Compute quotient of two real scalars, with overflow flag
27 F06BLF-D
28
29      SLATEC      GAMS:  A3
30
31 XADD-S      To provide single-precision floating-point arithmetic
32 DXADD-D      with an extended exponent range.
33
34 XADJ-S      To provide single-precision floating-point arithmetic
35 DXADJ-D      with an extended exponent range.
36
37 XC210-S      To provide single-precision floating-point arithmetic
38 DXC210-D      with an extended exponent range.
39
40 XCON-S      To provide single-precision floating-point arithmetic
41 DXCON-D      with an extended exponent range.
42
43 XRED-S      To provide single-precision floating-point arithmetic
44 DXRED-D      with an extended exponent range.
45
46 XSET-S      To provide single-precision floating-point arithmetic
47 DXSET-D      with an extended exponent range.
48
49      mssl3      GAMS:  A3
50
51 BRENT-S      Multiple precision arithmetic package by R. P. Brent.
52
53 EXRPACK-S      A package of subroutines that facilitate the use of
54 DEXRPACK-D      extended-range arithmetic.
55
56      NAG      GAMS:  A4
57
58 A02ABE-S      Modulus of a complex number
59 A02ABF-D
60
61 A02ACE-S      Quotient of two complex numbers
62 A02ACF-D
63
64 C06GBE-S      Complex conjugate of Hermitian sequence
65 C06GBF-D

```

```

66
67 C06GCE-S      Complex conjugate of complex sequence
68 C06GCF-D
69
70 C06GQE-S      Complex conjugate of multiple Hermitian sequences
71 C06GQF-D
72
73 C06GSE-S      Convert Hermitian sequences to general complex sequences
74 C06GSF-D
75
76 F06CLE-S      Compute quotient of two complex scalars, with overflow flag
77 F06CLF-D
78
79              SLATEC              GAMS:      A4
80
81 CARG-C        Compute the argument of a complex number.
82
83              SLATEC              GAMS:      A6
84
85 R9PAK-S       Pack a base 2 exponent into a floating point number.
86 D9PAK-D
87
88 R9UPAK-S      Unpack a floating point number X so that  $X = Y \cdot 2^{**N}$ .
89 D9UPAK-D
90
91              NAG                  GAMS:      A7
92
93 C06BAE-S      Acceleration of convergence of sequence, Shanks' transfor-
94 C06BAF-D      mation and epsilon algorithm
95
96 C06GBE-S      Complex conjugate of Hermitian sequence
97 C06GBF-D
98
99 C06GCE-S      Complex conjugate of complex sequence
100 C06GCF-D
101
102 C06GQE-S      Complex conjugate of multiple Hermitian sequences
103 C06GQF-D
104
105 C06GSE-S      Convert Hermitian sequences to general complex sequences
106 C06GSF-D
107

```

Here is what MenuDriver displays when this very shallow tree is expanded from the main menu. Note that all but the root node are terminal nodes, so none are marked as expandable. On the other hand, all level 1 nodes contain information that can be "shown", so are marked with "s" in column 3.² This tree contains no information nodes.

```

x   A      Arithmetic
  s   3      Real
  s   4      Complex

```

² Every terminal node must be a "show" node, but nonterminal nodes may also be "show" nodes. This would be indicated by "x s" in the display. (See the menu example in Chapter 1.)

```

s 6      Change of representation (incl. prime factorization)
s 7      Sequences (e.g., convergence acceleration)
eXpand label / Info. label / Show label / Main / Previous / Help / End :

```

If the user types "s 3", the following menu is displayed. Observe how this corresponds to the MTABLE information for this GAMS code.

```

s 3      Real

Index Library      Number of      Available
                  Programs      this system

1:   NAG            1            no
2:   SLATEC         6            yes
3:   mssl3          2            yes

1 / 2 / 3 / All / Return / Help / End :

```

The labels at the right are generated from the information on line 1 of menu, which is

```

18      SLATEC  MSSL  MATHLIB mssl3  PMATH

```

on this platform. Since NAG does not appear in the list, the NAG line is marked "no". The prompt invites the user type the index number of one of the listed libraries, or "a" to view all nine descriptions. Typing "3" produces the following:

```

mssl3      routines in this category

BRENT-S      Multiple precision arithmetic package by R. P. Brent.

EXRPACK-S    A package of subroutines that facilitate the use of
DEXRPACK-D   extended-range arithmetic.

```

Another library / Return / Help / End :

Note that the first routine exists only in single precision, whereas both single and double precision versions of the second routine are available. Typing "a" here will re-display the menu for this category, where another library may be selected. Typing "r" or simply hitting the "return" key will return to the original display, where another category can be selected. If the user now types "s 6", the following menu is displayed.

```

s 6      Change of representation (incl. prime factorization)

Index Library      Number of      Available
                  Programs      this system

1:   SLATEC         2            yes

1 / All / Return / Help / End :

```

This is somewhat redundant, since there is only one library that contains routines in this category. Typing "a" here will produce the following display. "1" would give the same information, but a different prompt.

```

SLATEC      routines in this category

```

```

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

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

```

Return / Help / End :

2.5.3. Contents of COMMON

In addition to the common variables that are set up by GENTREE from the first part of the database (see Advisor description), MenuDriver uses the following declarations:

```

      INTEGER LPTXT(200), NRTNS(200), NAVLIB, NLP
      COMMON /MDRCOM/ NLP, LPTXT, NRTNS, NAVLIB
C
      CHARACTER*8 GCODE(200), LIBNM(200), AVLIB(9)
      COMMON /CHRTR1/ GCODE, LIBNM, AVLIB

```

Most of these variables are read from the MTABLE of this database by GENTREE and are described above. The other two variables are set up from information read from the first line of menu by subroutine MMENUD as follows:

```

NAVLIB  number of available libraries. (Limited to at most 9.)
AVLIB   names of the available libraries. (No particular ordering is assumed, but they
        must be in the same case as they appear in the MTABLE.)

```

Caution: Because of the declaration for GCODE, only GAMS codes that have been truncated to eight or fewer characters can be handled by the present MenuDriver. In order to allow deeper expansion of some of the GAMS trees, this declaration would need to be changed wherever it appears. It seems most reasonable to use CHARACTER*24, since that is what is allowed for labels in the Advisor block CHRCTR. More seriously, the format of MTABLE, as read in GENTREE, would also have to be modified to allow more than eight characters. (Because of the change in database format, there would no doubt be some implications for the tools that are used to update the databases, as well. See Section 3.5.)

2.5.4. How It Works

Help processing. Help is available at various points in MenuDriver. In each case, HELPS2 is called with its first argument equal to the name of the help screen that is most relevant at this point. See Section 2.3 for details on how HELPS2 works.

Prompts and response processing. MenuDriver generates prompts and processes responses in exactly the same way as Advisor (see Advisor description). The only real difference in the main code is that it calls LEVSM instead of LEVSA.

Main menu processing. The only differences in main menu processing between Advisor and MenuDriver is that in the latter case MMENUD reads additional information

from line 1 of `menum` and sets up COMMON variables `NAVLIB` and `AVLIB`. In addition to processing the first part of the database, `GENTREE` reads the `MTABLE` information from the database into COMMON and leaves the file positioned at the first line of description text (line 24 in the example).

Screen display. Subsequent screens are displayed by `LEVSM`. Its basic operation is the same as `LEVSA` (see Advisor description). It allows the user to type one of the commands “x” (expand), “i” (information), “p” (previous), “s” (show) or “m” (return to the main menu).³ The “expand”, “information”, and “previous” commands are handled exactly the same as for Advisor, except that `PRTND` prints an “s” in column 3 of “show” nodes. It calls `SRCH3` to search `MTABLE` for this information.

Routine description display. The primary difference between Advisor and MenuDriver is in the processing of the “show” command. `LEVSM` first calls `SRCHR` to verify that the requested node exists; it then calls `SRCH3` to verify that it is a “show” node. If these tests are successful, `SROUT1` is called to display the descriptions available at the node.

`SROUT1` starts by calling `SRCHG` to determine which lines contain descriptions of routines in the selected category using the `MTABLE` information, display a menu of library availability information, and prompt the user for a selection. The part of the database file containing routine descriptions can be arbitrarily long and is not retained in memory. Instead, a buffer `IBUFF` of only 23 lines is used to contain the part that will be displayed in one screenful. As with the decision tree display, the only tricky part of `SROUT1` is determining exactly which lines are to be displayed so that descriptions are not split between screenfuls. When it is done, `SROUT1` rewinds the file and calls `GENTREE` to again position the file at the beginning of the description text.

2.5.5. Platform Dependencies

The program itself is standard Fortran 77. There are three key variables whose values may be platform-dependent:

<code>NMGDIR</code>	the path to the NMG directory structure.
<code>HFLST</code>	the path to the <code>mdrlist</code> file.
<code>HFDIR</code>	the path to the <code>helpfiles</code> directory, which contains the helpfiles.

`MAINM`, the MenuDriver main program, is instrumented to use Joe Grcar's `CHANGE` program to control `NMGDIR`.⁴ The other two variables are defined in terms of `NMGDIR`.

³ Note that “x” (expand) and “s” (show) generally have quite different meanings here. By convention, the user may type either “x” or “s” to show the information at a terminal node.

⁴ This is documented in the source code. Refer to Section 2.7 for more information about the use of `CHANGE`.

2.6. The Fetch Script

The NMG `fetch` command is implemented by a C shell script referred to as the Fetch Script. This script is actually called `fetch4`, for historical reasons. `fetch4` must be initiated with exactly one argument, assumed to be the name of the NMG statistics file, to which it appends detailed information on what has actually been fetched.

2.6.0. Overview

The script has a variable `prefix` that defines the directory which contains the script and the necessary files and directories to support this function. The contents of this directory are assumed to be:

<code>fetch4</code>	the NMG fetch script.
<code>doc/</code>	on-line documentation files.
<code>legal/</code>	legal notices.
<code>src/</code>	source files.
<code>super/</code>	the "superfiles" required for fetching complete source.

In addition to the various archive files described below, subdirectories `doc` and `super` also contain associated list-files that are examined to determine the existence of a requested document or source code. If `name` is one of these archives, the associated list-file is created via

```
ar t name > namelst
```

The script itself is divided into an initialization phase and a fetch phase. The latter is broken into two sub-phases, one for documentation fetching and the other for source code fetching. Its operation is made case-insensitive by converting responses to lower case before examining them. The script checks whether a file exists and warns the user before overwriting it. In case of a Fortran source, a response of "no" aborts the fetch; otherwise, it just skips the write.

The basic structure of the script is fairly simple, but it is complicated by the large amount of error coding, which checks for missing or unreadable files, etc. `fetch4` starts by asking the user for `rtn`, the name of a routine to be fetched. In order to account for the common mistake of entering the hyphenated type-designator that appears after most routine names in the MenuDriver descriptions, any hyphen-delimited extension is removed from the user-supplied name.¹ The script then asks what library contains `rtn` and enters the initialization phase. After completing the fetch phase, the user is asked whether another routine is to be fetched.² If so, the script cycles to the `rtn`-input stage, but the initialization phase is skipped (via script variable `refetch`) for subsequent routines.

¹ It might also be helpful to delete any ".f" extension from the typed name, but this would need to be done carefully to avoid compromising the fetching of non-Fortran files.

² The script is restricted to fetching from a single library only. The user must terminate `fetch4` and issue another `fetch` command to change libraries.

2.6.1. Help Facility

`fetch4` has a very rudimentary help facility. At the point at which it asks for a routine name, a response containing a question mark will cause the display of a built-in summary of the operation of the script. Such a response to the library name inquiry will cause the script to display a list of the libraries available for fetching on this platform. At these points, the user may also type "q" to terminate.

2.6.2. Initialization Phase

The initialization phase is executed exactly once per invocation of `fetch4`.³ The key script variables are all initialized to "none"; then follows a large switch statement to set values appropriate to each valid library. There may also be a one-time display of an important message about this library. The default case allows correction of a mistyped library name. The end of the initialization phase contains an extensive set of checks to verify that all required files exist for fetching from the selected library. At this point, any necessary list-files are created if they do not already exist. After successful completion, the user is informed whether documentation, source code, or both is available for this library. If both, the user is given of the choice of fetching either or both types of information for each routine requested from this library.

2.6.3. Documentation Fetching

Documentation fetching is quite straightforward. If `libdoc` is the name of the documentation archive for this library, then file `libdoc1st` is examined for the existence of the requested name, `rtn`. If present, `rtn` is extracted from the archive and its name changed to `rtn.doc`.⁴

A documentation archive is simply an `ar`-format file containing individual routine documents. For each routine documented, the name of the documentation file in the archive is the same as the routine name (lower case).

2.6.4. Source Code Fetching

Because a source code archive generally contains one routine per file, but we wish to deliver a complete source code for the requested routine, source code fetching is rather more complicated. There is a source archive, which is another `ar`-format file containing individual source files, with the name of the file the same as the routine name (lower case). In addition there must be an associated superfile archive and legal notice.

³ This is not quite accurate. If an invalid library name is typed, the list of acceptable names is displayed and the user is given a chance to type a new name and re-enter the initialization phase.

⁴ More sophisticated logic might be employed to avoid antagonizing users. As written, if a subsidiary routine is requested, one is first asked whether to fetch documentation. If the user says "yes", he/she is then informed that no document exists and is asked whether to fetch the source code.

Superfiles. "Superfiles" are text files containing the complete call tree for each routine for which source code is available.⁵ They permit fetching a complete source code (including all subsidiary routines) without the user having to know the underlying structure. The first line of a superfile contains the name of the routine itself. 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 (the "trivial superfile").

For an example, consider the SLATEC routine PCHIM. It has direct calls to PCHST and XERMSG, the SLATEC error handler. The superfiles for these three routines are listed below:

PCHIM	PCHST	XERMSG
pchim	pchst	xermsg
fdump		fdump
ilmach		ilmach
j4save		j4save
pchst		xercnt
xercnt		xerhlt
xerhlt		xerprn
xermsg		xersve
xerprn		xgetua
xersve		
xgetua		

We will not go into the detailed structure of XERMSG, but the above illustrates that is not sufficient for PCHIM's superfile to contain only the directly called routines if we wish to deliver a complete source code.

The superfiles are maintained in another ar-format file. For each routine with fetchable source code, the name of its superfile is the same as the routine name (lower case). With the exception of user callable entries defined in C-coded modules, there is a one-to-one correspondence between files in a source archive and those in the associated superfile archive. We illustrate this exception by displaying the superfiles for the PMATH routines CV16TO64 and CV64TO16, which are defined in the C-coded module `pmath_cnv.c`:

CV16TO64	CV64TO16	pmath_cnv.c
pmath_cnv.c	pmath_cnv.c	pmath_cnv.c
pm_params.h	pm_params.h	pm_params.h
pm_cnvset.h	pm_cnvset.h	pm_cnvset.h

Note that the superfiles for CV16TO64 and CV64TO16 are copies of the superfile for `pmath_cnv.c`. The latter also includes the names of two header files that it requires. `pm_params.h` and `pm_cnvset.h` have trivial superfiles which are not listed here. (If `pmath_cnv.c` required other C-coded modules, their names would also be included in

⁵ The origin of the term "superfile" has long been forgotten.

this superfile.) The PMATH source archive contains a file `pmath_cnv.c`, which is the actual source file, but it does not contain files named `cv16to64` or `cv64to16`.

How It Works. If `libsupsup` is the name of the superfile archive for this library, then file `libsupsup1st` is examined for the existence of the requested name, `rtn`. If present, `rtn` is extracted from the superfile archive. After a certain amount of validity checking, each routine contained in superfile `rtn` is extracted from `libsupsup`. If these are Fortran source files (no file extension), they are catenated together, with the requested routine first and the rest in alphabetical order, and the result called `rtn.f`. Other files are written exactly as extracted from `libsupsup`, and the user is informed how many files have been written and their names.

Legal Notices. In order to convey ownership and distribution information, a legal notice is appended to the beginning of each source file delivered by `fetch4`. (This does not apply to non-Fortran files with extension other than `.c`.)

2.6.5. Platform Dependencies

The script itself is pretty much standard UNIX. There are numerous script variables whose values may be platform-dependent:

<code>prefix</code>	the path to the <code>fetch</code> directory structure (assumed to contain subdirectories <code>/doc</code> , <code>/src</code> , <code>/super</code> , <code>/legal</code> , containing the documentation, source, superfile, and legal notice files, respectively).
<code>liblist1</code>	list of locally maintained libraries available via <code>fetch</code> .
<code>liblist2</code>	list of proprietary libraries with documentation available via <code>fetch</code> .
<code>archex</code>	archive file extract command.
<code>archls</code>	archive file list command.
<code>libdoc</code>	the path to the documentation archive for the current library.
<code>libsrc</code>	the path to the source code archive for the current library.
<code>libsupsup</code>	the path to the superfile archive for the current library.
<code>legal</code>	the path to the legal notice file for the current library.

`fetch4` is instrumented to use Joe Grcar's CHANGE program to control such platform-dependent details.⁶

2.6.6. Exit Codes

`fetch4` has a fairly extensive set of error return codes:

0	normal exit.
1	no document found.
2	no superfile found.
3	both of the above.

⁶ These are documented in the `fetch4` script itself. Refer to Section 2.7 for more information about the use of CHANGE.

- 4 necessary file missing or unreadable.
- 5 libsup or legal is "none" when libsrc isn't.
- 6 routine name not in its own superfile.
- 7 superfile is OK, but routine is not in source file.
- 8 inconsistent list file (routine in list, but not archive).

If any nonzero exit code is returned, in addition to an error message to the user, an appropriate entry is made in the statistics file. At present, the Main Control Script does not check for nonzero exit codes.

2.6.7. Usage Statistics

For each routine requested, `fetch4` writes one line to the statistics file (script variable `stat`) for each type of information (documentation or source code) requested. This will contain the routine name, library name, and what was done (including possible error messages).

All possible messages to the statistics file are given in the following list:

"Normal" messages:

```
Viewed $rtn document from $lib
Aborted $rtn document from $lib7
Saved $rtn document from $lib
Retrieved $file for $rtn source from $lib
Aborted $rtn source from $lib7
Fetched $rtn source from $lib
```

"Error" messages, indicating possibly mistyped name:

```
Error: no $rtn document in $lib
Error: no $rtn source in $lib
```

"Trouble" messages, indicating some problem with one or more Fetch files:

```
Trouble: libdoc = $libdoc bad for $lib
Trouble: libsrc = $libsrc bad for $lib
Trouble: libsup = $libsup bad for $lib
Trouble: legal = $legal bad for $lib
Trouble: $libdoc inconsistent with ${libdoc}lst ($rtn)
Trouble: $rtn superfile bad for $lib
Trouble: $rtn in super but $first not in src for $lib
Trouble: $libsup inconsistent with ${libsup}lst ($rtn)
```

In addition to the script variables defined above, this list contains:

```
rtn        the name of the routine being requested.
lib        the name of the library being fetched from.
first      the first entry in the superfile for rtn.
```

⁷ Written when user opts not to overwrite existing file.

2.7. The CHANGE Program

This section describes the use of CHANGE to maintain different versions of NMG scripts and source codes.

2.7.1. Basic Information

CHANGE is a portable Fortran program written by Joe Grcar (SNLA) to manage platform- or application-dependent change blocks within a single Fortran source code or C shell script. [1] The NMG Main Control Script and Fetch Script are instrumented to use CHANGE, as are some of the Fortran routines in NMGLIB. (Refer to earlier sections of this chapter for details.)

2.7.2. Instrumenting a Program for CHANGE

All that is required to instrument a program to use CHANGE is to include specially formatted comment lines before and after blocks of code that may need to be different in different contexts. The format for a Fortran source code is:

```
C*****LAB1
    < Code specific to block LAB1 >
C*****END LAB1
C*****LAB2
    < Code specific to block LAB2 >
C*****END LAB2
```

The lines between a pair of matching C*****-lines is called a "change block". The character strings "LAB1" and "LAB2", above, are called the "labels" for the change blocks. The comment character "C" can be replaced by "*", if desired. In a C-shell program, the comment character should be "#". A block that has a comment character in column 1 of all lines is declared "inactive".

Rules:

1. The line that terminates a block must be identical to the initial line except that "END " (note the space) is inserted after the fifth asterisk.
2. No other lines may begin with a comment character followed by five asterisks.
3. An active block must contain at least one non-comment line. (Thus, a change block cannot consist entirely of comments.)
4. There may be more than one block with the same label. All blocks with the same label must be active or inactive together.
5. Change blocks may *not* be nested.

The following is a code fragment from the Fetch Script to illustrate the use of change blocks:

```
# Set archive file extract and list commands:
#*****ar-format archives (recommended)
#       set archex = "ar x"
#       set archls = "ar t"
#*****END ar-format archives (recommended)
#*****tar-format archives
#       set archex = "tar xf"
#       set archls = "tar tf"
#*****END tar-format archives

# Set platform-dependent chain-name prefix:
#       prefix = location of fetch directory structure:
#               /doc  /src  /super  /legal
# First set default:
#       set prefix = /usr/local/math/nmg/fetch
#
# Overwrite only if necessary:
#*****AIX.OCFKMS
#       set prefix = /home/u08/pang_nmg/math/nmg/fetch
#*****END AIX.OCFKMS
```

There are three change blocks in this fragment, with labels "ar-format archives (recommended)", "tar-format archives", and "AIX.OCFKMS". Only the first of these is active. Note that whatever text follows the five asterisks will be displayed when CHANGE is run.

2.7.3. An Example

In the Section 2.2.3 we indicated that it is necessary to use CHANGE on source file `levs.f` in order to generate the source codes for routines LEVSA and LEVSM. Following is a terminal session for creating file `levsa.f`. It is assumed that the CHANGE source code (located in directory `nmglib/tools` of the `nmg.tar` file¹) has been compiled as

```
f77 -o change change80.f2
```

and that a copy of the executable change has been put in the directory which contains `levs.f`. As usual, information typed by the user has been formatted in **boldface**.

```
milstein% change
```

```
CHANGE:  VERSION 2.02 OF AUGUST 1992 BY JOSEPH GRCAR.
```

```
ENTER THE NAME OF THE ORIGINAL FILE.
```

```
FILE?  levs.f
```

```
375 LINES
```

```
STATUS OF THE CHANGE BLOCKS.
```

	ORIGINAL	COPY	NAME OF THE CHANGE BLOCK
	-----	-----	-----
1)	--	--	Advisor block

¹ See Section 2.8.

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`.

```

2)      --      --      Mdriver block

      ENTER THE NUMBERS OF BLOCKS TO CHANGE,
      OR ENTER BLANK TO ACCEPT THINGS AS THEY ARE.

NUMBER?  1

      ORIGINAL  COPY      NAME OF THE CHANGE BLOCK
      -----  -
1)      --      ACTIVE  Advisor block
2)      --      --      Mdriver block

      ENTER THE NUMBERS OF BLOCKS TO CHANGE,
      OR ENTER BLANK TO ACCEPT THINGS AS THEY ARE.

NUMBER?

      ENTER THE NAME FOR THE COPY,
      OR ENTER BLANK TO QUIT.

      levs.f (NAME OF THE ORIGINAL FILE)

FILE?  levs.a.f

      375 LINES

      FINISH.

```

Note that the display before the first "NUMBER?" prompt indicates that `levs.f` contains two change blocks, labeled "Advisor block" and "Mdriver block", and that both are inactive. In order to create the LEVSA source code, we tell CHANGE to activate the Advisor block by typing "1". The code then indicates that it has indeed made that block active in the copy. One can change the status of other blocks at the second "NUMBER?" prompt, if desired. Here we typed a carriage return to indicate acceptance of the changes, and CHANGE asked us to type the name to be used for the copy. (It must be different than the name of the original file.)

To create the LEVSM source code, we would run CHANGE again on `levs.f`, but this time type "2" at the "NUMBER?" prompt to activate the Mdriver block. We would then save the copy as `levsm.f`.

References

- [1] Grcar, J. F., *The Change Tool for Changing Programs and Scripts*, Version 2.02, Sandia Report SAND92-8225, Sandia National Laboratories, Livermore, California, August 1992.

2.8. How to Install NMG on a New Platform

The purpose of this section is to review the steps needed to install NMG on a new platform.

2.8.0. Basic Information

The complete set of files needed to install NMG on a new platform is provided in a tar-file called `nmg.tar`. Unfortunately, NMG has not reached the state where an automated installation process is possible. `nmg.tar` includes complete, if somewhat cryptic, installation instructions. This section is an attempt to clarify those instructions.

Because many of the files that support NMG are quite large, individual tar-files in various directories have been compressed. Otherwise it has proved impossible to find enough disk space available to even uncompress `nmg.tar.z`. As the instructions indicate, individual files should be uncompressed and converted in place.

2.8.1. Installation Overview

The complete installation of NMG involves the following steps:

1. Create a new directory and copy `nmg.tar` to it. This should be the final directory in which the NMG file structure will reside. (Otherwise, it may be necessary to insert temporary "installation" blocks into the Main Control and Fetch Scripts to point to this directory during the debug phase.) Execute "`tar xf nmg.tar`" to set up the full directory structure. Be sure to read file `README` in each directory. You will probably want to create a work directory `work/` in each.
2. Create a `CHANGE` executable and use it to process scripts and `NMGLIB`. (See below for more details.)
3. Install the Help Processor (`help/hlpc`). (See below for more details.)
4. Install Advisor Program (`advisor/adv`). (See below for more details.)
5. Install MenuDriver Program (`mdriver/mdrv`). (See below for more details.)
6. Set up ar-files in `fetch/subd/` (`subd = doc, src, super`) as directed in the associated `README` files. Update legal notices in `fetch/legal`, if necessary. (See below for more details.)
7. Update the contents of `help/helpfiles` as appropriate for your platform. Files `libabb` and `libava` are the ones most likely to require changes. If any help screens are added or deleted, you will also need to update the associated list-file, as well. (See Section 2.3 for more information.)
8. Create an `nmgnews` file for your platform. (Test it by executing "`newnmg news`".) We recommend that this be updated whenever anything in NMG is changed. Entries should probably be in reverse chronological order.
9. Set up an appropriate group to protect the contents of the NMG installation directory from unauthorized access. Update `nmgdr.c` as necessary to point to this directory

and compile it via "cc -o nmg nmgdr.c". The executable nmg should be installed in /usr/local/bin or other suitable public directory. (See Section 2.1.5 for more information.)

10. Update the manual page nmg.1 as necessary for your platform and install it in /usr/local/man or other suitable public directory.

Some of these steps are described in more detail in the following subsections.

2.8.2. Processing Scripts and NMGLIB

Before performing this step, it is necessary to decide where the NMG file structure will ultimately reside on your platform. If this does not correspond to any of the standard choices in the distribution file, you will need to add new change blocks to the newnmg and fetch4 scripts and to the NMGLIB source files main*.f.

Refer to Section 2.7 for information on what CHANGE does and how to compile it. The source code is located in directory nmglib/tools of nmg.tar. The first task is to insure that the appropriate change blocks are active in the scripts newnmg and fetch4 (in subdirectory fetch/).

Second, perform the following steps to create subdirectory nmglib/nmgsubs:

```
cd nmglib1
uncompress nmgsups.tar.Z
tar xvf nmgsups.tar
```

Then put a copy of the change executable in nmglib/nmgsubs and use it to create levsa.f and levsm.f, as described in the Section 2.7.3. The files main*.f could also be processed at this time, although this step can be deferred to the actual installation of the associated program if you do not wish to make "permanent" changes in the original source files.

2.8.3. Installing the Help Processor

To make and install the Help Processor, help/hlpc:²

```
asmb1 hlpsrc1st help.f nmgsups
f77 -o hlpc help.f3
```

¹ This first command assumes that you are starting at the top level of the NMG installation directory. If you have just created change in nmglib/tools, this becomes:

```
cp change ..
cd ..
```

² This assumes that you are starting in subdirectory nmglib/. If CHANGE was not run on mainh.f during step 2, it should be run on help.f before compilation. Note: since the changed copy must have a different name, one should do a "mv help.f help.f~" and tell CHANGE that help.f~ is the original file.

³ 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. (This also applies to the other two installation tasks, and will not be repeated there.)

The resulting files `help.f` and `hlpc` should be moved to `../help/work`, along with a copy of the list-files from `help/`, and tested there. One can simply type "`hlpc`" to start with the NMG general overview. Be sure that all screens listed in the help menu are readable and display properly. When OK, install in `help/` via:

```
cp hlpc ..
```

2.8.4. Installing the Advisor Program

Distribution directory `advisor/` contains `advisor.tar.Z`, a compressed tar-file which contains `menua`, `intree*`, and (possibly) `advnews`. It should be uncompressed and untarred. The file `advisor.lst` is the output from "`tar tvf advisor.tar`" on the NMG master machine.

To make and install the Advisor Program, `advisor/adrv`:⁴

```
cd ../nmglib
assmbl advsrc1st adriver.f nmgsups
f77 -o adrv adriver.f
```

The resulting files `adriver.f` and `adrv` should be moved to directory `../advisor/work`, with a copy of `menua` and `intree*`, and tested there. One can simply type "`adrv`" to start Advisor. Be sure that each of the main menu trees can be read and try out all commands at each level. Exercise each of the help screens available from the help menu. (To conserve disk space, delete `advisor.tar` and `advisor.lst` after verifying that `adrv` can successfully access and display from all `intree`-files.) When OK, install in `advisor/` via:

```
cp adrv ..
```

If there exists a file `advnews` in `advisor/`, then the NMG main control script will "more" it before starting up `adrv`. This can contain any short message you want users to see before executing the Advisor code.

2.8.5. Installing the MenuDriver Program

Distribution directory `mdriver/` contains `mdriver.tar.Z`, a compressed tar-file which contains `menua`, `intree*`, and (possibly) `mdrnews`. It should be uncompressed and untarred. The file `mdriver.lst` is the output from "`tar tvf mdriver.tar`" on the NMG master machine. Check the contents of the first line of file `menua` to make sure that it contains only libraries that exist on this platform. (Note that `MSSL`, `mssl3`, `PMATH`, and `SLATEC` are provided as part of `nmg.tar`.) See Section 2.5.1 for the format of this line.

⁴ This assumes that you are starting in subdirectory `advisor/`. If `CHANGE` was not run on `maina.f` during step 2, it should be run on `adriver.f` before compilation. Note: since the changed copy must have a different name, one should do a "`mv adriver.f adriver.f~`" and tell `CHANGE` that `adriver.f~` is the original file.

To make and install the MenuDriver Program, `mdriver/mdrv`:⁵

```
cd ../nmglib
asmb1 mdrsrc1st mdriver.f nmgsups
f77 -o mdrv mdriver.f
```

The resulting files `mdriver.f` and `mdrv` should be moved to directory `../mdriver/work`, with a copy of `menua` and `intree*`, and tested there. One can simply type "mdrv" to start MenuDriver. Be sure that each of the main menu trees can be read and try out all commands at each level. Exercise each of the help screens available from the help menu. (To conserve disk space, delete `mdriver.tar` and `mdriver.lst` after verifying that `mdrv` can successfully access and display from all `intree`-files.) When OK, install in `mdriver/` via:

```
cp mdrv ..
```

If there exists a file `mdrnews` in `mdriver/`, then the NMG main control script will "more" it before starting up `mdrv`. This can contain any short message you want users to see before executing the MenuDriver code.

2.8.6. Preparing the Fetch Archives and Legal Notices

The archive files in `fetch/` subdirectories `doc/`, `src/`, and `super/` are provided as compressed `tar`-files. This is because we have found that the format of `ar`-files is not completely portable between different UNIX platforms. These will be needed as `ar`-files during the operation of the Fetch Script.⁶ Thus, all `tar.z`-files need to be uncompressed and converted to `ar`-format via script `tartoar`, which will be found in directory `nmglib/tools/` of `nmg.tar`. The following procedure should be followed in each of these directories:

```
foreach name (`echo $list`)
    uncompress ${name}.tar.Z
    tartoar ${name}.tar $name
    ar tv $name > ${name}.arlst
end
```

The contents of `$name` should be compared with `${name}.lst`, the output from "tar tvf `${name}.tar`" on the NMG master machine, before removing `${name}.tar`. The relevant information (size, time stamp, filename) is contained in the last six fields of the "tv" listing. For the provided listings, these can be extracted via

⁵ This assumes that you are starting in subdirectory `mdriver/`. If `CHANGE` was not run on `mainm.f` during step 2, it should be run on `mdriver.f` before compilation. Note: since the changed copy must have a different name, one should do a "mv `mdriver.f` `mdriver.f~`" and tell `CHANGE` that `mdriver.f~` is the original file.

⁶ We have found that extracting from `ar`-format archives is much faster than using `tar`-format on many platforms. If you wish to retain the original `tar`-files, use `CHANGE` on the `fetch4` script to activate the `tar`-block and deactivate the `ar`-block. You will also need to drop ".tar" from the file names.

```
awk '{ printf("%6s %3s %2s %5s %4s %s\n", $4, $5, $6, $7, $8, $9) }' \
${name}.lst > ${name}.lsta
```

A similar listing, possibly with different field numbers, depending on the exact format of the "ar tv" listings on your platform, can be generated from file `${name}.arlst`. If all is well, `diff` will reveal no differences between these files.

Caution: On platforms with limited disk space, it may be necessary to convert each library independently, rather than all at once as indicated above.

The appropriate lists are as follows:

directory	file list
fetch/doc/	mathdoc mssldoc nagdoc pmathdoc sladoc ⁷
fetch/src/	mssl mssl3 pmath slatec
fetch/super/	super3 superm superp supers

Notice: The machine constants in files `d1mach`, `i1mach`, and `r1mach` in `fetch/src/slatec` may need to be modified so that the constants for the machine on which NMG is being installed are active (non-commented). There must be exactly one set of active constants in each of these routines. The Sun constants are active in the distribution file.

Subdirectory `fetch/legal/` contains legal notices needed by the Fetch Script. These are simple text files that are concatenated to the source file. These may need minor editing, depending on the platform. In particular, note that `legals` contains a line about machine-dependent constants. This needs to be consistent with the active constants in the above-mentioned SLATEC routines.

Once all of these directories have been set up, you should exercise the `fetch4` script extensively.⁸ In particular, you should try to fetch documentation and source code for at least one routine from each library, to make sure that the right links have been established in the script and that each file is readable. In the case of PMATH, try fetching both a Fortran routine (such as `SZERO`) and an entry to a C-coded module (such as `RANF8`). Examine the extracted files for reasonableness. (Then delete them!)

2.8.7. Validation and Clean-up

Once all of the testing that has been indicated in the previous subsections has been carried out, you should delete all of the `tar`-files and their associated listings, to conserve disk space. You can also delete the entire contents of subdirectory `nmglib/`. (This can be recreated, if necessary, via `"tar xvf nmglib.tar nmglib"`.)

You should then exercise all of the available commands, first by executing `newnmg` directly and then via the executable `nmg` created in step 9. (This need not be an exhaustive

⁷ If the NAG Online Supplement is not licensed for this platform, you *must* delete `nagdoc` and deactivate the "NAG available" block in `fetch4`.

⁸ Remember that `fetch4` takes an argument (call it `stat`, say) and that it must be reinitiated for each different library.

test of each function, but just enough to insure that the links in newnmg have been set up correctly and that each command is operational.) It may take some experimentation (and local modifications to newnmg) to get `mail` and the usage statistics function to work properly.

At this point you should carefully examine the file permissions in the entire NMG directory structure. All executables, scripts, and directories (including the top level) should have permissions 770 (read-write-execute for owner and group only); all archives and data file should be 660. Having done this, it is a good idea to have someone not in the group try to execute NMG's commands.

If you have access to the NMG Validator (see Chapter 4), it would also be a good idea to do a complete validation run, to verify internal consistency of the archives and databases you have installed.

2.8.8. Tool Descriptions

Following are brief descriptions of the tools that are included in `nmg.tar` directory `nmglib/tools`.

Program name: `artotar`

Type: C shell script

Purpose: Converts an ar-format file to an equivalent tar-format file.

Usage: `artotar arfile [tarfile]`

Input: `arfile`, the ar-format file to be converted.

Output: `tarfile`, an equivalent tar-format file (`arfile.tar`, if not specified).

Notes: An attempt is made to preserve time stamps in the conversion process.

Program name: `assmbl`

Type: C shell script

Purpose: Assembles a collection of files from a directory into a single file.

Usage: `assmbl namefile [outfile [dir]]`

Input: `namefile`, a file containing the list of files to be collected.

`dir`, the directory from which files are to be read (current working directory, if not specified).

Output: `outfile`, the resulting assembled file (assembled, if not specified).

Notes: Files will be assembled in the order in which the names appear in `namefile`. A message will appear on `stdout` if a file mentioned in `namefile` is absent. Refer to Section 2.2.3 for a list of standard `assmbl` arguments and other information.

Program name: `CHANGE`

Type: Fortran program (source code: `change80.f`)

Purpose: This is Joe Grcar's `CHANGE` program, which is useful in maintaining platform-dependent sections in Fortran source files and C-shell scripts.

Notes: Refer to Section 2.7 for compilation instructions and a usage example.

Program name: comparem

Type: C shell script

Purpose: Compares the contents of two directories.

Usage: comparem *dir1 dir2* > *compout*

Input: *dir1* and *dir2*, the two directories to be compared.

Output: *compout*, the resulting comparison. (Goes to stdout if not redirected.)

Program name: tartoar

Type: C shell script

Purpose: Converts a tar-format file to an equivalent ar-format file.

Usage: tartoar *tarfile* [*arfile*]

Input: *tarfile*, the tar-format file to be converted.

Output: *arfile*, an equivalent ar-format file (*tarfile.a*, if not specified).

Notes: An attempt is made to preserve time stamps in the conversion process.

Program name: updarf

Type: C shell script

Purpose: Update an ar-format archive from a directory of modified files.

Usage: updarf *arfile dir* > *updarf.log*

Input: *arfile*, the ar-format archive to be updated.

dir, a directory containing one or more possibly updated *arfile* files.

Output: *arfile*, the updated archive file.

updarf.log, a log file indicating which files have been updated.

Notes: updarf only works properly if *arfile* contains only text files. The object is to replace only files which have been updated, leaving the time stamps on the others unchanged.

2.9. Notes on Future Development

This final section contains notes on desirable future developments for the various components of NMG. Some are very specific suggestions and others are more general. Most of these items would likely be taken care of if the present system were replaced by an HTML-based system.

2.9.1. General

- There are a couple of places where the assumed number of lines per screen are built into the databases. This number is also hard-wired into several of the NMGLIB routines. We should prepare for eventual user control over this parameter, assuming that this control is kept in the Fortran codes at all. (See next suggestion.)
- Use more (or other standard UNIX command) for display of screens, rather than the specially-written routines in Help, Advisor, MenuDriver. (This will make it easier to adapt to the user's environment.)
- Because of problems with different versions of a routine in different libraries, routines being in different GAMS categories, etc., it seems more reasonable to keep a master database with one record per routine. Only the master database would be updated directly. Special tools would be used to extract Advisor or MenuDriver databases from the master database. This would enable us to implement a **locate** command, such as was available in the original NMG, set up synonym tables, etc. It would also provide a means for supplying test codes and other associated materials, when available. A master database would also have the advantage that a change in format for the Advisor or MenuDriver databases would only require modifying the tool that extracts them from the master database.

2.9.2. Help

- Subroutine HFDISP could be made both simpler and more effective by invoking more rather than doing its own display processing. (Check: Would we lose anything in the process?)

2.9.3. Advisor

- Some redundant code could be eliminated from LEVSA (file `levs.f`) if the SRCHR call and associated tests were moved before the code that selects among the "x", "s", and "i" options.
- In order to be more useful in an environment in which not all libraries are available on all machines, Advisor should provide separate recommendations for different libraries, with the preferred one specially marked somehow.
- We should provide an easy way to get to MenuDriver or **fetch** from Advisor. (Xnmg had a prototype of how this might work. An HTML-based system would probably be easier to maintain.)

2.9.4. MenuDriver

- See Advisor for possible LEVSM changes.
- Subroutine SROUT1 could be simplified (and probably speeded up) if the MenuDriver descriptions were in a separate file, say `mtable1`, instead of in `intree1`. This would likely simplify database maintenance procedures, too.
- As above, we should provide an easy way to get to **fetch** from MenuDriver .

2.9.5. Fetch

- We should provide for cross-library fetching (e.g., a routine in MSSSL needs to use a routine in SLATEC or PMATH).
- We should provide a facility for synonyms, related routine fetching, etc. (Master database?)
- We should make the set-up less Fortran-specific.

2.9.6. `nmg.tar`

- We should devise a more fool-proof procedure to make sure that the `nmg.tar` archive is kept up-to-date.