1989

A general purpose menu processor user interface tool for mechanical engineering applications /

Catherine Mary Curtin
Lehigh University

A GENERAL PURPOSE MENU PROCESSOR USER INTERFACE TOOL FOR MECHANICAL ENGINEERING APPLICATIONS

by

Catherine Mary Curtin

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Masters in Science

in

Mechanical Engineering

Lehigh University

1989

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering.

May 16, 1989

Date

Tuluğa Ozroy

Professor in Charge

F. Erdogan

Chairman of Department

ACKNOWLEDGEMENTS

I would like to thank Dr. Tulga M. Ozsoy, my graduate advisor, for his ideas and enthusiasm throughout the development of this thesis. Also, I would like to thank Joe Clifford, my husband, for convincing me to pursue a graduate degree.

TABLE OF CONTENTS

CERTIFICATE OF APPROVAL ii

ACKNOWLEDGEMENTS iii

TABLE OF CONTENTS iv

LIST OF FIGURES vii

ABSTRACT 1

CHAPTER 1. Introduction 2

1.1 Introduction 2

1.2 User Interface 3

1.3 Implementation 5

1.4 Terminologies 5

CHAPTER 2. Implementation 7

2.1 Introduction 7

2.2 Automatic Implementation 10

2.3 Interactive Implementation 11

2.4 VAX Operating System Installation 14

CHAPTER 3. Features 15

3.1 Introduction 15

3.2 Built-in Global Menu 16

3.2.1 Dialogue Style Device 16

3.2.2 Unigraphics Device 17

3.3 Application Dependent Global Menu 17

3.3.1 Global Menu Name 18

3.3.2 Unigraphics Device 18

3.3.3 Menu Dependency 19

3.4 Messages 19

3.5 Type Ahead 21

CHAPTER 4. Data File Specification 24

4.1 Introduction 24

4.2 Menu Data Set 24

4.3 Option Delimiters 25

4.4 Pointers 26

4.4.1 To Submenus 26

4.4.2 To Subroutines 27

4.5 Application Dependent Global Menu Information 27

4.6 Message Data Set 28

4.7 CONTROL.FOR 28

CHAPTER 5. Program Structure 31

5.1 Introduction 31

5.2 Menu Initialization 31

5.2.1 Initialization Routines 31

5.2.2 Reading Data 32

5.3 User Interface 36

5.4 Processing User Input 38

5.5 Global Menus 38

5.5.1 Built In Global Menus 38

5.5.2 Application Dependent Global Menus 39

CHAPTER 6. UIT$CREATE 40

6.1 Introduction 40

6.2 File Access 40

6.3 Menu Database 41

6.4 Create Menu Branch 41

6.5 Change Menus 42

6.5.1 Change Option 42

6.6 Messages 43

6.7 Global Menu 43

6.8 List Hierarchy 44

6.9 Save File 44

6.10 Program Structure 45

6.11 Installation on a VAX 45

REFERENCES 47

APPENDIX A - Menu Processor Routines 49

APPENDIX B - UIT$CREATE Routines 64

APPENDIX C - Sample Tree Structure Data File 90

APPENDIX D - Sample UIT$CREATE session 95

VITA 106

LIST OF FIGURES

Figure 2.1. File Relationships. 8

Figure 2.2. Sample main program for automatic menuing. 11

Figure 2.3. Sample main program for interactive menuing. 13

Figure 3.1. Built-in global menu. 17

Figure 3.2. Redefining global menu name. 18

Figure 3.3. Sample subroutine using message information. 21

Figure 3.4. Sample subroutine accessing type ahead buffer. 22

Figure 4.1. General menu data set format. 25

Figure 4.2. Option line pointing to a submenu. 27

Figure 4.3. Option line pointing to a subroutine. 27

Figure 4.4. Example menu title data line. 28

Figure 4.5. Example message data line. 28

Figure 4.6. Trivial data file. 29

Figure 4.7. Associated CONTROL.FOR. 29

Figure 5.1. Sample MENDAT and MID arrays. 34

Figure 5.2. ACCESS array. 35

Figure 5.3. KEY_CHARS and KEY_ID arrays. 35

Figure 6.1. File Access menu. 40

Figure 6.2. Menu Database menu. 41

Figure 6.3. Create Menu Branch menu. 42

Figure 6.4. Change Menu menu. 42

Figure 6.5. Change Option menu. 43

Figure 6.6. Messages menu. 43

Figure 6.7. Global Menu Status menu. 44

Figure 6.8. Sample menu hierarchy list. 44

Figure 6.9. Save File menu. 45

Figure A.1. Menu processor common blocks. 50

Figure B.1. UIT$CREATE common blocks. 64

ABSTRACT

This thesis discusses a general purpose menu processor user interface tool for applications in Mechanical Engineering. The menu processor triggers subroutines or submenus based on user input to an applications program.

The menu processor relies on a data file for menuing hierarchy information. To build an interactive application program, the menu processor trigger function library is linked with applications subroutines. The menu processor may be implemented to control an entire application or the application programmer may choose to interface with the processor on a subroutine by subroutine basis.

The menu processor is written in FORTRAN on a VAX/VMS[1] operating system. The user interface supports two types of devices, Tektronix 41-series[2] terminals and workstations with a program function keyboard (PFK) for user input.

The menu processor includes several features in addition to the menu hierarchy triggering function. These include built-in and applications dependent global menuing, user option selection type-ahead, and warning and error message storage. Also, the data file containing the hierarchy information may be automatically generated using UIT$CREATE, an example of an interactive application using the menu processor.

---

[1]VAX/VMS is a trademark of Digital Equipment Corporation.

[2]Tektronix 41-series is a trademark of Tektronix, Incorporated.

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

Many computer applications in mechanical engineering require the use of menus to guide the user through an interactive software program. A user selects options from a list and the menu processor forces the appropriate action to follow based on a particular menu hierarchy. Currently, each programmer must write a menu processor to fit his application. A common general purpose menu processor would alleviate this duplication of effort and free the programmer to concentrate on the applications.[3] In addition, if implemented within a company or a department, both programmers and users would enjoy a standard user interface.[4]

This thesis discusses a general purpose menu processor user interface written for the mechanical engineering community at Lehigh University. It is written in FORTRAN on the VAX/VMS operating system. It supports VT100[5] dialogue display style terminals, Tektronix 41-series terminals, and Unigraphics II[6] workstations. The

---

[3]Kirk Christenson, "Writing Easy-to-Use Programs for Computers," *Mechanical Engineering,* Volume 4 No. 12, pp 66-69.

[4]Dan R. Olsen, William Buxton, Roger Ehrich, and David J. Kasik, "A Context for User Interface Management," *IEEE Computer Graphics and Applications,* Volume106 No. 9, pp. 33-41.

[5]VT100 is a trademark of Digital Equipment Corporation.

Tektronix terminals and Unigraphics workstations support graphics applications.

1.2 USER INTERFACE

A common menu processor user interface can alleviate many applications programming problems. First and foremost is the duplication of effort warranted for programmers without this tool. Each programmer must decide how to attack the user interface problem and then spend time implementing it. Also, because developing a user interface is considered application dependent, the interface is not often versatile.[7] For example, the programmer must alter sections of existing code in order to make small additions or modifications to his user interface.

A common menu processor provides a more consistent interface to both users and programmers. If the processor is easy to integrate into an application, programmers will opt to incorporate it into applications on a regular basis. The menu processor design should allow modifications or additions to the menu structure to be quick and simple with no rewrite of the programmer's original code.[8] Error or warning messages also should be processed separately from the application code for easy changes and foreign language portability. The users benefit from a common user interface by being familiar with the features of the interface independent of the application[6]. This

---

[6]Unigraphics II is a trademark of McDonnell Douglas Manufacturing and Engineering Systems Company.

[7]Olsen, Buxton, Ehrich, and Kasik, pp.34-38.

[8]Christenson, pp. 66-69.

eradicates much of the unusability of current applications by users other than the programmer.[9] In addition, the user will regard applications programs with a common interface as much more unified.

Two user interfaces of the menu processor support the two types of devices. Both the VT100 display and the Tektronix 41-series terminals are dialogue style devices. In this interface, the menus scroll along the left side of the screen. From these menus, the user selects an option by keying in the alphanumeric characters corresponding to the characters displayed before a delimiter for that particular option. This type of environment is offered by the IDEAS[10] Solid Modeling and Design software.[11] The other interface supports the workstations with two separate devices, a message monitor (MM) and a program function keyboard (PFK). . In this case, the menus appear on the MM and the user select an option by pushing the button on the PFK corresponding to the number of the option on the menu. This type of environment is offered by the Unigraphics II software.[12] The Tektronix 41-series terminals may also be used as Unigraphics workstations where the dialogue area is used as a message monitor. If an application supports both types of devices, the programmer may specify that option

---

[9]Robert F. Sproull, W. R. Sutherland, and Michael K. Ullner, *Device-Independent Graphics,* (New York: McGraw-Hill Book Company, 1985), pp. 206-210.

[10] IDEAS is a trademark of Structural Dynamics Research Corporation.

[11]Structural Dynamics Research Corporation, *IDEAS User's Guide,* (Milford OH: Structural Dynamics Research Corporation, 1988), pp. 11-1,11-3.

[12]McDonnell Douglas Manufacturing and Engineering Systems Company, *Unigraphics II Design Module,* (Cypress CA: McDonnell Douglas Manufacturing and Engineering Systems Company, 1987), pp. 1-3.1,1-3.3.

numbers are displayed on the dialogue style interface, thus the application appears device independent.

1.3 IMPLEMENTATION

The general purpose menu processor consists of several parts. It relies on a tree data structure file for all menu interaction data. A library of routines drive the menuing according to its menu hierarchy. In addition, each application must have its library of routines that are triggered by menu selections.

The programmer may choose from two separate types of implementation. The first, and simplest, is for the programmer to make a single call to the menu processor triggering function and allow the processor to drive the entire application. The other method allows the programmer to make individual calls to the processor on an as needed basis.

The general purpose menu processor relies on a tree structure data file. According to specification, this file holds the information on menus, the relationships among menus and applications subroutines, additional information on global menu status, and warning or error messages. An interactive menu-driven program allows the programmer to create a menu hierarchy and save the data to a file that conforms to specifications.

1.4 TERMINOLOGIES

Tree structure refers to the menu hierarchy. Pointers refer to the interaction of elements within the menu hierarchy. For example, a portion of a tree structure may be that the third option of the second menu points to the fifth menu. The uppermost

menu in the hierarchy is called the top level menu or main menu.

The programmer implements the menu processor into his application. This application, in turn, is intended for the user. The menu processor and its related files reside in the user interface tool (UIT) directory on a VAX. For example, to create a data file, the programmer runs the UIT$CREATE program.

Chapter 2

IMPLEMENTATION

2.1 INTRODUCTION

There are two implementations of the general purpose menu processor. In the automatic implementation, the applications program triggers the interaction handler function at a specified level of the menu hierarchy. Branching down the hierarchy and activating menus from the menu library or subroutines from the applications library is controlled wholly by user selected options. In the interactive implementation, the applications program activates the menu triggering function one level at a time. Most applications prefer a mixture of these two implementations.

In general, the programmer creates an applications library, a main program, a tree structure data file, and an automatically created controlling subroutine. He links these modules together with the menuing trigger function library to obtain an executable image of the main application program.

The tree structure data file includes a field that indicates whether an option points to a subroutine or a menu, as well as a field to indicate which menu or subroutine the option calls. Appendix C lists a sample tree structure data file. Chapter 3 discusses the tree structure data file specifications in detail. The programmer may create the data file according to specifications or he may run the UIT$CREATE program in order to do this for him. Chapter 6 discusses the UIT$CREATE program in detail.

The programmer's first step is to create a library of his applicatons subroutines.

For the purposes of this document, APP.OLB will designate this applications library file. Second, the programmer sets up a menu structure to guide his application and he creates a menu tree data file. Even if he created this file according to specification, he must run UIT$CREATE once to write the subroutine CONTROL.FOR from the data file. This subroutine performs all of the necessary calls to the applications library based on user selected options. In addition, CONTROL.FOR must not be renamed. An example CONTROL.FOR is listed in Appendix D. Finally, he must construct the main program, MAIN.FOR, which activates the menuing. Figure 2.1 outlines these relationships.

Input	Into	Output
Interactive Data	UIT$CREATE	MENU.DAT CONTROL.FOR
MAIN.FOR CONTROL.FOR UIT library Application library	Link command	MAIN.EXE
MENU.DAT	MAIN.EXE	Application program

Figure 2.1. File relationships.

In general, the arguments to the pre-processor subroutines are the data file name, the delimiter, and the display switch. If the data file field sent is blank, the

menu processor assumes the name MENU.DAT as the default. Each menu option has an associated set of key characters that fully specify the option for selection. The delimiter is the character that separates the key characters from the remainder of the option description. The characters that fall before the delimiter are the key characters. For example, in the following option:

ST-Calculate stresses

the delimiter is a dash - and the key characters are ST. If no delimiter is defined, the processor assumes that the first four characters in an option are the key characters. The switch variable determines wheter the key characters or option numbers will be displayed on a menu. If the switch variable equals 1, then the options are displayed as they are found in the data file, complete with delimiter. If the switch variable equals 2, then the options are displayed with option numbers while the key characters are ignored. For example, with a switch variable of 2, the above option would appear:

4 Calculate stresses

if it were the fourth option on the menu. A switch variable of 2 is recommended in an application that is implemented on both the dialogue style device and the Unigraphics workstations so that menu selection appears device independent to the user. The switch integer is automatically assumed equal to 2 when linking with the Unigraphics device menu trigger function library. The data file variable must be declared CHARACTER*30; the delimiter variable CHARACTER*1; and the switch variable INTEGER*2.

Before the programmer may run his application, he must link all of the compiled modules together with the menuing trigger function library. The order of the modules in the link command is important. The Digital Command Language (DCL)[13]

linking commands for the dialogue style terminals and the Unigraphics workstations follow respectively:

```
$ LINK MAIN, CONTROL, APP/L, UIT$MENUS/L
$ LINK MAIN, CONTROL, APP/L, UIT$UFMENUS/L, UGUSER2/L
```

These result in one executable file, MAIN.EXE, the executable image of the applications program.

2.2 AUTOMATIC IMPLEMENTATION

The automatic implementation requires that each menu option points to either a subroutine or a menu. The menu tree structure data file contains all of the pointer information. The main application program issues a single call to the MENUS subroutine of the menuing trigger function library. The applications programmer then links the modules together and runs the resulting executable image. During the application program execution, if the user chooses an option that points to a submenu, the submenu is displayed. If the user selects an option that points to a subroutine, the subroutine is executed and the original menu remains active.

The main program issues a call to the menuing trigger function library's MENUS routine. The arguments for this subroutine are the tree structure data file name , the delimiter symbol, the first menu activated, and a display switch. The first menu activated variable must be declared CHARACTER*30. Figure 2.2 lists a sample main program.

---

[13]DCL is a trademark of Digital Equipment Corporation.

```
      PROGRAM MAIN
C     This is provided by the applications programmer
      CHARACTER*30 NAME, FIRST
      CHARACTER    DEL
C     Any initialization goes here
      NAME  = 'THESIS.DAT'
      FIRST = 'Top Level Menu'
      DEL   = '-'
C     Invoke the menus
      CALL MENUS (NAME, DEL, FIRST,1)
      END
```

Figure 2.2. Sample main program for automatic menuing.

2.3 INTERACTIVE IMPLEMENTATION

The interactive implementation attends to the cases where more versatility is warranted. For example, an option first executes a subroutine, then branches to another menu. Self-contained modules that can be classified as separate from the rest of the application, such as view manipulations, may use this implementation also. The interactive implementation may be used in conjunction with the automatic implementation.

In order to invoke a menu, a call must be issued to the menu trigger function library TREE routine. The arguments are the starting menu, the option selected, the next menu based on the option selected, and the display switch. The starting menu variable and the display switch comprise the input variables. The option selected variable and the next menu variable comprise the output from the subroutine. After TREE has returned the next menu branch and the option selection via key characters,

the applications program acts on this information. The menu processor is only activated during a call to TREE. The starting menu variable and the next menu variable are declared CHARACTER*30. Each one must contain the exact spelling, including capitalization, as the corresponding menu in the data file. The option selected variable must be declared as CHARACTER*4.

There are two methods of using the TREE subroutine. The more common use drives the entire application from the start. The other mixes some calls to TREE with the original call to MENUS as in the automatic case. This second method is recommended when only a few special cases exist. In both methods, the menuing trigger function routines manage automatically the simple cases where an option points to either a subroutine or a submenu based on the tree structure data and the CONTROL subroutine. After the simple case has been handled, control of the application returns to the application program rather than to the menu processor.

In the first method, the program must make a call to INIT_MENU in order to read in the data file and set up the arrays. The arguments of this subroutine are the data file name and the delimiter symbol. When a particular option points to both a subroutine and a menu, the data file pointer must be to the submenu. After an option has been selected, TREE returns the key characters of the option in the option selected variable. It also determines the next menu in the hierarchy and returns this information via the next menu variable. The applications program checks the option returned and acts upon it accordingly. The TREE subroutine may be called from anywhere in the program, including within applications subroutines. This approach is recommended for very large applications because of the potential complexity of the menus.

The example program in figure 2.3 illustrates this method. The option whose

key character are ST requires both a subroutine call and a branch down the menu hierarchy. The data file specified that this option points to a submenu, returned in the NEXT variable. Also, the application begins at the "Read in file" menu.

```
      PROGRAM APPLIC
      CHARACTER*30  FILE, START, NEXT
      CHARACTER*4   STATE
      CHARACTER*1   DEL
      FILE   = 'THESIS.DAT'
      DEL    = '-'
      NEXT   = 'Read in File'
      CALL INIT_MENU (FILE, DEL)
50    START = NEXT
      CALL TREE       (START,  STATE,  NEXT, 1)
      IF (STATE. EQ. 'ST')  CALL STRESS
      GO TO 50
      END
```

Figure 2.3. Sample main program for interactive menuing.

Alternatively, when there are relatively few instances of an option pointing to both a menu and a subroutine, some minor adjustments to the automatic method are in order. The main program is identical to the main program of the automatic implementation as in Figure 2.3. In this case, the data file pointer indicates the subroutine rather than the menu. The call to TREE indicates that a menu is also warranted. Within the subroutine TREE is called with the START variable equal to the next menu down the hierarchy. At this point, however, the subroutine does not return any of the TREE data to the automatic part of the application. Therefore, the subroutine must use the TREE subroutine for all control beyond this point in the

hierarchy.

2.4 VAX OPERATING SYSTEM INSTALLATION

The menuing trigger function libraries must be installed by the system manager on a VAX system. The user interface tool kit directory must be created in which the menus libraries reside. All files within this directory need a world read and world execute protection. Then, a system logical name must be defined for each library. The directory is created with the following DCL commands:

```
$ CREATE/DIR  [UIT]
```

The libraries MENUS.OLB and UFMENUS.OLB are installed with the following DCL commands:

```
$ DEFINE/SYS  UIT$MENUS     UIT$MENUS.OLB
$ DEFINE/SYS  UIT$UFMENUS   UIT$UFMENUS.OLB
```

These logical name definitions should be included in the system startup command procedure. Once these logical names are defined, all users on the system may access the libraries for linking purposes.

Chapter 3

FEATURES

3.1 INTRODUCTION

Several features distinguish the automatic menu processor user interface tool other than automatic menuing according to a hierarchy. Most of these features are modeled after those found in the computer-aided engineerig graphics packages Unigraphics II and IDEAS. A main feature is the option selection. On the Unigraphics II user interface, the user selects options by pushing lit buttons on a PFK while the menu is listed on the MM. [14] On the IDEAS interface, the user selects options by typing in key characters of each menu item listed on the screen. [15]

A feature common to both systems is the built-in global menu. This menu is available to the user for selection at all times. In Unigraphics, however, some of the global options are not available to a menu when not feasible.

IDEAS allows the user to type several option key characters on one command line. If an error is encountered, IDEAS aborts reading the remainder of the command line. This type ahead capability is extremely useful for the user who is very familiar with the package. [16]

---

[14] Unigraphics II Design Module, p. 1-3.1.

[15] IDEAS User's Guide, p. 1-4.

3.2 BUILT-IN GLOBAL MENU

The built-in global menu is an inherent part of the general purpose menu processor. These global options are available at all times on all applications that use the menu processor. There are three menu control options in the global menu, reject, main menu, and list global menu. The "reject" option brings the user up one level in the menu hierarchy. The maximum levels that the hierarchy may have is 100. The "main menu" option moves the user to the top level menu regardless of the current position. The "display global menu" option lists the global menu options.

3.2.1 DIALOGUE STYLE DEVICE

Because the menus scroll on the left side of the screen, there are three additional built-in global options for dialogue style devices. The first is "clear screen" which erases all dialogue from the screen. The second is "write menu" which rewrites the current menu on the dialogue area. The third is "menu display" which toggles the menu display between on and off. Figure 3.1 is a listing of the built-in global menu.[17]

---

[16]IDEAS User's Guide, p. 1-5.

[17]IDEAS User's Guide, p. 1-8.

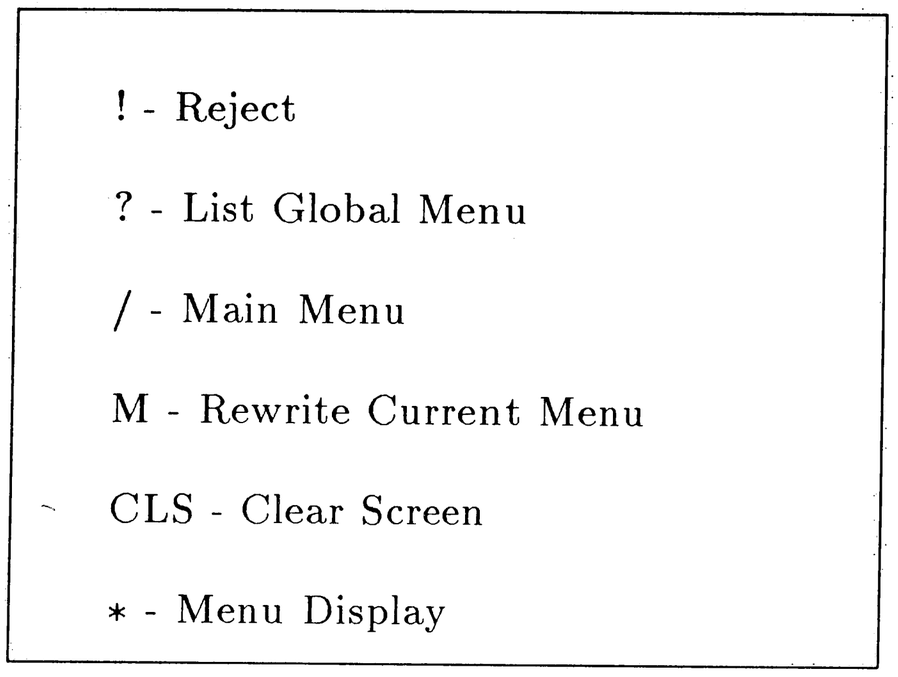


Figure 3.1. Built-in global menu.

3.2.2 UNIGRAPHICS DEVICE

On a Unigraphics device, the user selects options by pushing lit buttons.[18] If a user selects an unlit button, nothing happens. Because of the MM, the "write current menu" and "clear screen" options are unneccesary. This reduces the built-in global menu to two options, each with a permanently set button. The "Reject" button activates the reject option. The "Terminate Operation" button serves as the main menu option.

3.3 APPLICATION DEPENDENT GLOBAL MENUS

The programmer may want to include more global menu options than have been built into the menu processor. Any menu in the menu data file is eligible to be a global menu. This menu name must be specified in the main program before the initialization call. In the global menu listing, the application dependent options follow

---

[18]Unigraphics II Design Module.

the built-in options. Every application dependent global option may be turned on or off for each menu so the programmer may control which global options are available to which menu. They are assumed off unless specifically turned on in the data file.

3.3.1 GLOBAL MENU NAME

The global menu name defaults to "Global Menu" unless otherwise specified. In order to specify a new name for the global menu, the programmer must include the GLOBAL common block in his main program and redefine the variable GMWORD. The menuing trigger function library searches for this menu in the data file and fills the global menu arrays accordingly. GMWORD is declared CHARACTER*30. Figure 3.2 lists a sample portion of a main program that redefines GMWORD.

```
PROGRAM MAIN
COMMON / GLOBAL/ GMWORD
CHARACTER*30 GMWORD
GMWORD = 'Change Parameters'
         :
```

Figure 3.2. Redefining global menu name.

3.3.2 UNIGRAPHICS DEVICE

To access the global menu on a Unigraphics device, the global menu button must be pushed. These devices have a limited number of buttons that may be implemented, so that the global button may change from menu to menu. It will always be the last button in a menu list. Since the list is limited to fourteen choices and one is taken up by the global menu, the programmer should limit a menu that will be

implemented on a Unigraphics device to thirteen options per menu unless no application dependent global menu exists.

3.3.3 MENU DEPENDENCY

Each option on the applications dependent global menu may be activated differently for each menu in the application. This information resides in the menu tree structure data file. For example, an application has four additional global menu options. For the seventh menu, option 2 is turned off, while the other three are on. If the current menu is the seventh menu, and the user keys in ? to list the global menu, the following is listed to the screen underneath the built-in global menu:

F-First Option

T-Third Option

FT-Fourth Option

The second option of the global menu is unselectable at this time. On a Unigraphics workstation, these would be listed as the first three options under the global menu.

3.4 MESSAGES

Accessing warning, error, or informational messages is an additional feature to the user interface. Because the messages are stored in the data file rather than hard-coded into the application, minor changes to messages do not require recompiling and linking to create the executable image. This allows the programmer more flexibility in changing the syntax of his interface. He may choose to define his final output statements at one time by editing the messages in the data file, thus providing users with a more understandable interface.[19] Further, if the application is for the international market, the only alteration necessary to translate the application into

another language; i.e. Turkish; is to translate the information in the data file. With this in mind, it is not difficult to plan the application such that all output to the screen is stored in the tree structure data file.

In order to access a message, the programmer must include the INOUT MESSAGE and common blocks in his subroutine. The INOUT common block contains the input and output registers. The MESSAGE common block contains the WARERR array with all of the message information. It is declared CHARACTER*30 and has 1000 elements. The messages are stored in the sequential order in which they appear in the data file. For example WARERR(5) is the fifth message found in the message section of the data file.

The programmer may choose to mix the WARERR information with his data or use it alone. The INFORM subroutine lists a message to the screen according to message number. If the programmer needs to mix other information in the message, the subroutine LONG determines how many of the 30 characters are filled. Its input arguments are the message variable and the total number of characters. The output argument is the filled number of characters. Both of the integer variables are declared INTEGER*2. Figure 3.3 lists a sample subroutine using the message information.

---

[19]Christenson.

```
      SUBROUTINE EXAMPLE
      COMMON / MESSAGE / WARERR (1000)
      COMMON / INOUT     / IDINP, IDOUT
      CHARACTER*30 WARERR
      INTEGER*2        L1, L2
      CALL INFORM (4)
      X = 3.0
      CALL LONG (WARERR(6), L1, 30)
      CALL LONG (WARERR(7), L2, 30)
      WRITE (IDOUT,100) WARERR(6), X, WARERR(7)
100   FORMAT (5X,A<L1>,F4.1,1X,A<L2>)
      RETURN
      END
```

Figure 3.3. Sample subroutine using message information.

3.5 TYPE AHEAD

On a dialogue style device, an option is selected by typing in the key characters of that option after the "Enter Command: " request. Users familiar with an application may choose to key in several sequential option key characters at one time. These sequential commands are stored in the type ahead buffer.

The user may type in a list of commands up to 30 characters in length. This type ahead is applicable also in the interface where option numbers are displayed rather than key characters. Each command must be separated from the next with either a space or a comma. The command line is read as one unit and then decoded into elements of four characters or less. The menu processor then processes each command separately. On detecting an invalid option, the processor clears the entire buffer and the

remainder of the command line is aborted.

The programmer may want to extract user input from the type ahead buffer to his application. In order to access the buffer, the application subroutine must include the TAHEAD common block. Figure 3.4 lists a sample portion of an application subroutine accessing the type ahead buffer.

```
                SUBROUTINE APPLIC
                COMMON /  INOUT  / IDINP, IDOUT
                COMMON /  TAHEAD/ BUF, PBUF, MAXBUF
                CHARACTER*4  BUF(30),  WLEN
                INTEGER*2      PBUF, MAXBUF
                        :
C               Find length
                WRITE(IDOUT,*) ' Enter length:  '
50              PBUF  =  PBUF  +  1
                IF  (PBUF.GT.MAXBUF)  THEN
                   READ (IDINP,100)  LENGTH
                ELSE
                   WLEN  =  BUF (PBUF)
                   DECODE (WLEN,300,ERR=50)  LENGTH
                END IF
C               Continue application
                        :
```

Figure 3.4. Sample subroutine accessing type ahead buffer.

First the application subroutine declares the common blocks and the variables. PBUF is the pointer to the latest element of the BUF type ahead buffer. MAXBUF is the number of elements in the buffer. When the buffer is searched for input, the application increments the pointer and checks it against MAXBUF. Then the subroutine reads the

increments the pointer and checks it against MAXBUF. Then the subroutine reads the next element of BUF and decodes it from a character string to a numeric value. If the next element of BUF cannot be decoded into a value, execution returns to the line requesting user input.

Chapter 4

DATA FILE SPECIFICATION

4.1 INTRODUCTION

The tree structure data file contains all information regarding the menu hierarchy, global menu options, messages, and subroutine pointers. The file specification is fairly rigid. A file is most easily created or updated by running the CREATE program and electing to save the information. This also automatically generates the CONTROL.FOR subroutine. Appendix C lists an example data file.

A tree structure data file consists of a number of data sets. Each data set holds either menu data or message data designated by a type number. A type number 100 indicates that a menu data set follows; a type number 200 indicates that a message data set follows. Between each data set is a line containing the integer -1.

4.2 MENU DATA SET

A data set type number 100 designates that the following information composes a menu. The menu data set consists of a title followed by options. The menu data sets are separated from one another by -1 delimiters. The menu processor can handle a maximum of 1000 menus and 10000 total options per tree data file. Data fields to the right of the menu title or option hold any additional information such as pointers or global menu status. Both menu titles and menu options data fields have a length of thirty characters. There is no maximum number of options per menu but a maximum

of thirteen is recommended because only thirteen are in any one menu on a Unigraphics device. Figure 4.1 lists the general format for a menu.

```
100
Title of Menu
Option 1
Option 2
:
Option N
-1
```

Figure 4.1. General menu data set format.

The first menu listed in the data file must be the top level menu, however this is the only restriction on the order of menu data sets in the file. The top level menu is the trunk of the menu tree. Control is returned to the top level menu upon the user selection of 'main menu' from the global menu. Often, it is the first menu displayed in the application.

4.3 OPTION DELIMITERS

Delimiters help the user to determine exactly what key strokes are necessary to choose an option. If a delimiter is used in the options of a menu, it should be specified in every option within the data file for continuity among menus. These delimiters promote continuity from menu to menu. They also allow for less key strokes per option selection because if a delimiter is not used, the key characters for the option default to the first four characters in the option. For example, if a dash '-' is the delimiter, it appears in every option somewhere within the first five characters of the thirty character field.

4.4 POINTERS

Within an application, the selection of a menu option triggers some response. This response may be to display a new menu, to call a subroutine, or to execute some other code from the applications program. Regardless of the response, there must be a pointer in the data file to guide the menuing trigger function library to react properly.

The pointer field occupies the same line as the option. It is a 1 character integer field occupying the 31st character in the line. A 1 points to a menu; a 2 points to a subroutine; and a 0 or blank field indicates that the menuing trigger function library is not responsible for triggering a response on this option. A character field follow directly next to the integer pointer field. It is 30 characters long and indicates to which menu or subroutine the option points, depending on the integer field.

4.4.1 TO SUBMENUS

If a 1 occupies the pointer integer field, the option points to a submenu. The following character field names this menu. The spelling and capitalization of the submenu must match that of the corresponding menu's title. This menu must exist somewhere in the data file or the application program will encounter an error when this option is selected. The programmer may preprocess the data file to test for this compatibility by using the 'check file' option in the UIT$CREATE program. When using the interactive implementation, it is important to remember that if an option points to both a submenu and a subroutine, the submenu is to be named in the data file since the application will call the subroutine when the option is selected. Figure 4.2 shows an example option line of a data file pointing to a submenu.

```
PO- Specify Point          1Generic Point Menu
```

Figure 4.2. Option line pointing to submenu.

4.42 TO SUBROUTINES

If a 2 occupies the pointer integer field, the option points to a subroutine. The following character field names this subrutine. This field also includes any arguments of the subroutine. The CREATE program must be run at least once and the save option must be chosen in order to create the subroutine CONTROL.FOR which deals with these subroutines. Figure 4.3 shows an example option line of a data file pointing to a subroutine.

```
AB- Absolute Coordinates     2 ABSOLUTE
```

Figure 4.3. Option line pointing to subroutine.

4.5 APPLICATION DEPENDENT GLOBAL MENU INFORMATION

Each menu in the tree may specify which application dependent global menu options are available to it. Since the global menu is menu dependent, this information is included on the line containing the menu title. This data field begins in column 32 to correspond with the start of the pointer subroutine or submenu of the option lines. This field may accomodate up to 32 items in the application dependent global menu.

The data field is composed of 1's and 0's in sequence. A 1 signifies that an option is available; a 0 signifies that an option is unavailable. The first integer in the data field indicates the status of the first option on the global menu; the second integer

indicates the status of the second option on the global menu, and so on. If the field is blank, no options are available to the menu. This is useful when no application dependent global menu exists. Although this field may be up to 32 switches long, it need only accomodate as many options as the global menu holds. Figure 4.4 lists an example title data line for a menu to which the first, third, and fourth options of a five option application dependent global menu are available.

```
Create Geometry          10110
```

Figure 4.4. Example menu title data line.

4.6 MESSAGE DATA SET

A data set with type number 200 designates that the following information composes a message data set. There is a maximum of 1000 messages. Each message is in a field of 30 characters. The programmer accesses a message by its relative placement in the data file. For example, the tenth message in the data file is message #10. Figure 4.5 lists an example message data line.

```
Warning:  Surface not closed
```

Figure 4.5. Example message data line.

If a message is longer than 30 characters, it should be truncated at the 30th character and continued on the next data line in the file.

4.7 CONTROL.FOR

Once the tree structure data file is completed, the controlling subroutine CONTROL.FOR is created. CONTROL.FOR is unique for each data file. If one

option from a menu tree is deleted, CONTROL.FOR must be recreated. The most reliable way to create it is to run the CREATE program and save the information.

The CONTROL.FOR subroutine triggers the applications subroutines where necessary. It calls subroutines based on relative placement of the options within the file. The subroutines it calls are equivalent to the data field containing subroutine name after a subroutine pointer in the data file.

The relative placement parameter is equal to the placement of the option in the data file. Both menu titles and menu options are counted in this placement. For example, the trivial data file in figure 4.6 would produce the CONTROL.FOR in figure 4.7.

```
100
Top Level Menu
ST-Stresses                1First Menu
EX-Exit                    2EXIT
-1
100
First Menu
SH-Show Stresses           2STRESS
-1
```

Figure 4.6. Trivial data file.

```
SUBROUTINE CONTROL (NOPT)
IF (NOPT.EQ.3) CALL EXIT
IF (NOPT.EQ.5) CALL STRESS
RETURN
END
```

Figure 4.7. Associated CONTROL.FOR.

If the data file were changed so the 'EX- Exit' were no longer an option, CONTROL.FOR would also require changes. If the applications subroutines include arguments that must be commoned and/or declared, the programmer must edit the CONTROL.FOR file accordingly. Since the menu processor triggers the applications subroutines and control of the application alternates between the menu processor and the application, any arguments to these routines would be lost during the menu processor activation. Therefore, the use of common blocks in the application subroutine is recommended.

Chapter 5

PROGRAM STRUCTURE

5.1 INTRODUCTION

The general purpose menu processor program consists of three parts; menu initialization, user interface, and option processing. The menu initialization part sets up the menuing data structure. The device dependent user interface part displays the menus and reads user response. The processing part controls the application according to the user's input command and the menu hierarchy.

5.2 MENU INITIALIZATION

The menu initialization initializes variables, reads in the data file, and fills the menu hierarchy arrays. The initialization differs slightly between the automatic implementation and the interactive implementation. The programmer interface of the automatic implementation is simpler than that of the interactive implementation. In the automatic implementation, only one call is made to the menuing trigger function library to cover both the initialization and the menu processing.

5.2.1 INITIALIZATION ROUTINES

In the interactive implementation, the applications program must call INIT_MENU before proceeding with any menu processing. This routine initializes menuing variables and calls the routine to read the data file. The arguments for

INIT_MENU are the tree structure data file name and the delimiter symbol, respectively. Appendix A lists the calling sequence for INIT_MENU. Figure 2.3 lists a sample applications program using INIT_MENU.

Before issuing a call to read the data file, INIT_MENU initializes several variables; the input and output registers, the type ahead buffer pointers, and the reject pointer. If the programmer has not specified a data file name, it defaults to MENU.DAT. The global menu name is checked against the default "Global Menu", also. Then INIT_MENU calls READ_MENUS to read the menu data into menuing arrays.

In the automatic implementation, the applications program must issue a call to MENUS. This routine performs all of the initialization in INIT_MENU, as well as the menu processing. Its calling sequence is listed in Appendix A. Figure 2.2 lists an example program using this automatic implementation.

For the Unigraphics user interface, there is an additional initialization call in both INIT_MENU and MENUS. The User Function routine UF1000 is called to initialize the Unigraphics terminal. This call enables later calls to the User Function library.

5.2.2 READING DATA

The READ_MENUS routine fills the menuing arrays by reading in the tree structure according to data file specifications. It reads the menu data sets and message data sets of the specified data file and fills the menu and pointer arrays. Finally, it determines the key characters of each option. At the conclusion of the READ_MENUS routine, the initialization is complete.

READ_MENUS reads each line of the data file and interprets it. The first line of the data file indicates whether the data set immediately following is a menu data set or a message data set. After a data set has been read, the next indicator is read and the data set following is interpreted. This continues until the end of the data file is encountered.

If a message data set is encountered, the message arrays are filled. As each message is read and put into the WARERR array, the message counter, NWE, is incremented. These messages are available to the programmer via the MESSAGE common block and/or the INFORM subroutine. Figure 3.3 lists an example use of this.

If a menu data set is encountered, the menu arrays are filled. The four arrays filled directly from the data file are the MENDAT, MID, ACCESS, and GLOPT arrays. MENDAT contains the actual menus, as well as the submenu and subroutine interaction data. MID and ACCESS are pointer arrays. GLOPT contains application dependent global menu option data.

The MENDAT array holds the menu information in the sequential order of the data file. MENDAT is a two column array and stores up to 11000 rows of data. The menu titles and option s are stored in the first column of MENDAT. The subroutine or submenu to which an option points is stored in the second column of the array. In the rows containing a menu title in the first column, the second column is blank. MENDAT is declared CHARACTER*30.

The MID array describes the trigger function of an option for the menu processor. It is a one column array of the same length as MENDAT. The ith element of MID may be 0, 1, or 2. A 0 indicates that the menu processor ignores the option selected. A 1 indicates that the ith option in MENDAT points to a submenu. This

submenu is stored in MENDAT (i,2). A 2 indicates that the option points to the subroutine stored in MENDAT (i,2). MID is declared INTEGER*2. Figure 5.1 lists a sample MENDAT and associated MID arrays.

MENDAT		MID
View Manipulations		0
MO-Model	Model	1
S-Screen	Screen	1
Model		0
T-Translate	MTRANS	2
R-Rotate	MROT	2
EYE-Eye	MEYE	2
Screen		0
T-Translate	STRANS	2
R-Rotate	SROT	2
EYE-Eye point	SEYE	2

Figure 5.1. Sample MENDAT and MID arrays.

The ACCESS array holds the pointers to MENDAT and the key characters arrays. ACCESS is a three column array and stores up to 1000 rows of pointers. The ith line in ACCESS describes the ith menu in the MENDAT array. The first column of ACCESS points to the start of the ith menu in MENDAT. The second column is the number of options in the ith menu. the third column points to the beginning of the ith menu in the key character array. the key character array will be discussed later in this document. ACCESS is declared INTEGER*2. Figure 5.2 lists the ACCESS array for the MENDAT in figure 5.1.

ACCESS

1	2	1
4	3	3
8	3	6

Figure 5.2. ACCESS array.

After the data file has been successfully read, the two option selection arrays are filled. The KEY_CHARS array holds the key characters in sequential order of the options only. Because key characters may have a length from 1 to 4 characters, the KEY_ID array hold the length of each corresponding KEY_CHARS. KEY_CHARS is declared CHARACTER*4; KEY_ID is INTEGER*2. Figure 5.3 lists these arrays for the MENDAT of Figure 5.1.

KEY_CHARS	KEY_ID
MO	2
VP	2
T	1
R	1
EYE	3
T	1
R	1
EYE	3

Figure 5.3. KEY_CHARS and KEY_ID arrays.

In addition, the menu number of the application dependent global menu is determined. The titles of each menu are checked until a match is made with the GMWORD string variable. GMWORD defaults to "Global Menu" unless changed in

the applications program before the menu initialization is called as in figure 3.2.

5.3 USER INTERFACE

The first half of the TREE subroutine contains the device dependent user interface. The menu processor drives the entire user interface for dialogue devices. This includes menu display, option selection, and option validity checking. For Unigraphics workstations, User Function routines drive the interactive user devices for the interface. This includes message monitor (MM) and program function keyboard initialization, menu display on the MM, lighting the function buttons on the PFK, option selection through button pushes, and option validity checking.

For the dialogue style device, the menu processor displays the menu as well as handles the user response. In order for the dialogue interface to more closely resemble the Unigraphics interface, it is possible to display the options with option numbers rather than with the key characters and a delimiter. The user responds by keying in characters which are then checked against the available options. These responses must have the exact capitalization as listed in the menu. In addition, a user familiar with an application may choose to type several responses on one command line.

In the dialogue style interface, TREE first determines the menu number and then lists the menu to the screen. The MENNUM subroutine receives a character string containing the menu name and returns the number of the menu in the data structure. Next, DSPLY_MENU is called to list the menu to the screen. DSPLY_MENU utilizes the pointers in ACCESS to determine which portion of MENDAT to display. If the display switch variable of 2 is sent to DSPLY_MENU, the key characters and delimiter are stripped from an option and option numbers are displayed instead. At the end of

the menu listing, DSPLY_MENU requests the user to enter a command.

After the menu has been displayed, the user keys in his command. TREE reads the input from the user into a string variable. The string is broken into commands, delimited from each other by a space or a comma. These commands are stored in the type ahead buffer. The type ahead buffer is set to 1 to be ready to process the first command. After a command is processed, the type ahead buffer pointer is incremented for the next command.

In order to determine an option's validity, the current string in the type ahead buffer is checked against the key characters of the options in the current menu and in the global menu. The pointers in ACCESS are used to determine which elements of KEY_CHARS to search. If the user has input an invalid option, an error message is written to the screen, the type ahead buffer is cleared, and the user is prompted to input a new response.

In the Unigraphics style interface, after determining the menu number, TREE calls a User Function routine, UF1603, to display the menu, light the PFK according to the menu, and receive user feedback. The arguments are the menu title, the menu array, the length of the menu, the default setting, and the output user response.[20] The User Function library returns only valid button pushes so that the processor need not determine if a button were lit when it was pushed. Also, there is no type ahead concept with button pushes.

---

[20]User Function Manual, McDonnell Douglas Corporation, (1988), Cypress CA, p. 3.3.19.

5.4 PROCESSING USER INPUT

The processing part controls the menu triggering or applications subroutine calling based on user input. The processing is device and implementation independent. When the user interface part determines that the option is valid for processing, it also stores the valid option's relative position in the data arrays. The associated element in MID then determines whether a submenu is triggered or an applications subroutine is called. If a menu is to be triggered, the menu processor updates the current menu information to reflect the next menu in the hierarchy. If a subroutine is to be called, TREE calls CONTROL which was created based on the data file. After the processing is complete, TREE returns the original menu, the option selected, and the next menu in the tree structure.

5.5 GLOBAL MENUS

During processing, the valid option may be checked against the global menu. Both the built-in global menu and the application dependent menu are searched. The GLOBAL routine processes options found on the global menu.

5.5.1 BUILT- IN GLOBAL MENU

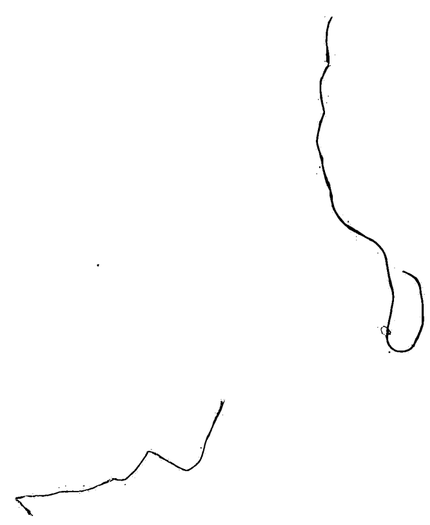
The built-in global menu contains six options; main menu, list menu, clear screen, toggle menu display, display global menu, and reject. Main menu changes the value of the menu number to 1. List menu lists the current menu to the screen, regardless of the menu display toggle. Clear screen clears all dialogue from the screen. Toggle menu display switches between "display menu" and "do not display menu". Display global menu lists the global menu including the applications dependent global

menu options. Reject moves the menu up one level in the hierarchy.

The reject buffer for the reject option is dynamic. As each new level is reached in the hierarchy, the reject buffer, REJECT, stores the new menu number and the pointer to the current level in the hierarchy. When the reject option is chosen, the pointer moves one level up in the hierarchy and the new menu number is obtained from REJECT. The REJECT buffer is cleared when the user chooses the main menu option.

5.5.2 APPLICATION DEPENDENT GLOBAL MENU

The GLOBAL subroutine also tests the option against the application dependent global menu. These options are checked in the same manner as they are against the current menu. The difference is that some global menu options may not be available to the current menu. For the ith menu, GLOPT (i) must be translated from an integer into a series of 1's and 0's. The GLVAL subroutine accepts the integer and returns an array of the 1's and 0's. An option is only checked if its corresponding switch is equal to 1.



Chapter 6

UIT$CREATE

6.1 INTRODUCTION

The UIT$CREATE program allows a programmer to interactively create or modify a menu hierarchy data file, as well as the associated CONTROL.FOR subroutine. To begin creating or modifying a menu hierarchy, one must run the UIT$CREATE image. UIT$CREATE is an example of an application of the general purpose menu processor using the interactive implementation. Appendix D lists a sample UIT$CREATE session.

6.2 FILE ACCESS

The first menu encountered in UIT$CREATE is the File Access menu listed in figure 6.1. This menu allows one to modify an existing tree structure from a data file or to begin a new tree structure. To modify or add to an existing data file, one chooses RE and specifies the file name. UIT$CREATE reads this file and updates its arrays before starting the session. To create a new data structure, one selects NEW. After initializing the session, UIT$CREATE branches to the top level menu.

```
File Access
RE- Read in existing file
NEW- Create new file
```

Figure 6.1. File Access menu.

6.3 MENU DATABASE

The top level menu is called Menu Database. All of the main functions are accessed from this menu listed in figure 6.2. From this, one may choose to create a menu, modify an existing menu, set the application dependent global menu status, add or modify warning or error messages, list the hierarchy, save the information to a file, or exit UIT$CREATE.

```
Menu Database
CR- Create new menu branch
CH- Change menu branch
GL- Global menu status
ME- Messages
LH- List hierarchy
SA- Save entire menu tree
EX- Exit
```

Figure 6.2. Menu Database menu.

6.4 CREATE MENU BRANCH

In order to create a menu data set, one selects CR from the top level menu. Next, one enters the title of the menu branch. Then one chooses options from the Create Menu Branch menu listed in figure 6.3. To add an option to the current menu, one selects AD. The options and associated pointers to subroutines or submenus are added in sequential order to the current menu. If the option added points to a submenu, the submenu must have the exact capitalization as in its menu title. To view the current menu and its pointers, one selects LI. To finish working on this menu, on uses reject (!) or main menu (/) from the built-in global menu to return to the Menu Database menu.

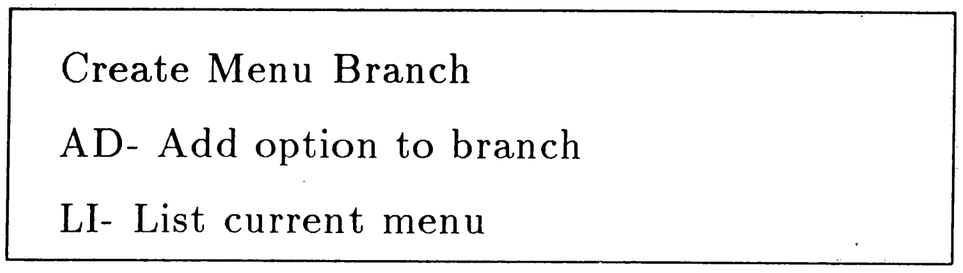


Figure 6.3. Create Menu Branch menu.

6.5 CHANGE MENUS

To modify an existing menu, one selects CH from the top level menu. This branches to the Change Menu menu listed in figure 6.4. To alter the title of a menu, one selects CT, chooses the menu to rename, and keys in a new title. To delete an entire menu data set, one selects DB and subsequently selects the menu to delete from a list. To list all of the menu names created, one chooses LM. In order to modify options within a menu, one selects CO.

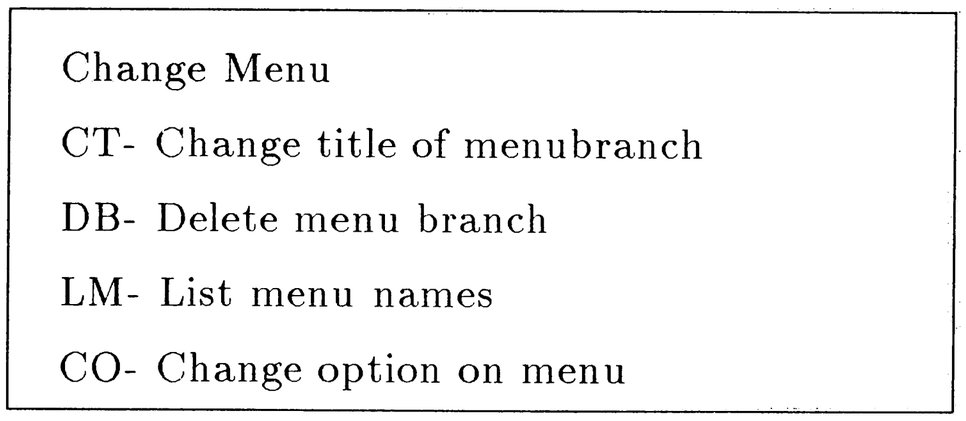


Figure 6.4. Change Menu menu.

6.5.1 CHANGE OPTION

After selecting which menu to modify, one chooses the modification type from the Change Option menu listed in figure 6.5. One selects AD to put an additional option on the menu. The new option follows all of the previous options in the menu. One chooses DE to delete an option from the menu. RE is chosen to replace a particular menu choice. LI lists the entire menu to the screen.

```
Change Option
AD- Add option to branch
DE- Delete option from branch
RE- Replace option from branch
LI- List current menu branch
```

Figure 6.5. Change Option menu.

6.6 MESSAGES

In order to create a message data set, one chooses ME from the top level menu. This branches UIT$CREATE to the Messages menu listed in figure 6.6. To add a warning or error message to the data, one selects AM. To delete a message, one selects DM and then deletes a message from the list. RE is selected to replace a message in the list with a new message. LM simply lists the messages sequentially.

```
Messages
AM- Add message to list
DM- Delete message to list
RE- Replace message to list
LM- List messages
```

Figure 6.6. Messages Menu.

6.7 GLOBAL MENU

To manipulate the application dependent global menu, GL is selected from the Menu Database menu. This branches to the Global Menu Status menu listed in figure 6.7. To add an option to the global menu, AD is chosen. This assumes that a default menu called "Global Menu" has been created or that one of the existing menus has been designated as the global menu. To select an existing menu as the application dependent

global menu, one chooses RE. This must also be declared in the applications program as in figure 3.2. To set the global menu switches of a particular menu, ST is chosen and then the menu is specified. The switches default to 1 or on until toggled off. LI lists the applications dependent global menu.

```
Global Menu Status
AD- Add option to global menu
ST- Set status of global menu
RE- Select menu to be global
LI- List global menu
```

Figure 6.7. Global Menu Status menu.

6.8 LIST HIERARCHY

To list the menu hierarchy that has been built, one chooses LH from the top level menu. If a starting menu is chosen from a list, the menu tree structure is displayed from that point down the branches. For each branch in the hierarchy, the menu name is indented to indicate a lower menuing level. Figure 6.8 lists a sample hierarchy list.

```
Create Geometry
    Generic Point
    Line Type
        Generic Point
    Arc Type
        Generic Point
```

Figure 6.8. Sample menu hierarchy list.

6.9 SAVE FILE

To save the menuing tree sturcture data to a file, SA is chosen from

the main menu. This branches to the Save Tree menu listed in figure 6.9. The CHK option checks that all of the menu data sets that are pointed to by an option have been created. For example, if an option points to a menu called "View Manipulations", the CHK option checks this menu exists in the data. The menus that are pointed to but do not exist are listed to the screen. One may choose to return to the main menu and create these menus. To save the data to a file, one opts for SA from the menu. This creates a menu tree structure data file and the associated CONTROL.FOR subroutine. If the data file has been created or modified using editing rather than UIT$CREATE, this step is necessary to produce the proper sequencing in CONTROL.FOR. Even a minor change to a single menu data set requires a new CONTROL.FOR.

```
Save Tree
CHK- Check tree
SA-  Save tree to file
```

Figure 6.9. Save File menu.

6.10 PROGRAM STRUCTURE

UIT$CREATE uses the general purpose menu processor to trigger its submenus and subroutines. It maintains a data structure of the input menu data parallel to those of the menuing trigger function library. The subroutines store this data in common blocks and manipulate it according to user input. The common blocks, purpose of each subroutine, and each subroutine's calling sequence are listed in Appendix D.

6.11 INSTALLATION ON A VAX

To install UIT$CREATE, the VAX system manager must have a copy of CREATE.EXE and CREATE_MENU.DAT in the UIT$ directory. They must have

world read and world execute protection. The system logical name UIT$CREATE is made with the following DCL command:

```
$ CREATE/SYS UIT$CREATE [UIT]CREATE.EXE
```

This should be included in the system startup command procedure. Then all users of the system may access to the UIT$CREATE program.

REFERENCES

Christenson, Kirk. "Writing Easy-to-Use Programs for Computers." Mechanical Engineering, Volume 106 No.9, 1983, pp.66-69.

Digital Equipment Corporation. Common Graphics Interface User's Manual, An Integration Architecture Module. Maynard, MA: Digital Equipment Corporation, 1987.

Digital Equipment Corporation. Programming in VAX FORTRAN. Maynard, MA: Digital Equipment Corporation, 1984.

McDonnell Douglas Manufacturing and Engineering Systems Company. User Function Manual. Cypress, CA.: McDonnell Douglas Manufacturing and Engineering Systems Company, 1988.

McDonnell Douglas Manufacturing and Engineering Systems Company. Unigraphics II Design Module. Cypress, CA: McDonnell Douglas Manufacturing and Engineering Systems Company, 1984.

Olsen, Dan R., Buxton, William, Ehrich, Roger, and Kasik, David J. "A Context for User Interface Management." IEEE Computer Grapichs & Applications, Volume 4 No. 12, (1984), pp. 33-41.

Sproull, Robert F., W. R. Sutherland, and Michael K. Ullner. Device-Independent Graphics. New York, NY: McGraw-Hill Book Company, Inc., 1985.

Structural Dynamics Research Corporation. Geomod Solid Modeling and Design. Milford OH: Structural Dynamics Research Corporation, 1988.

Structural Dynamics Research Corporation. IDEAS User's Guide. Volume 1, Milford OH : Structural Dynamics Research Corporation, 1988.

Appendix A

MENU PROCESSOR ROUTINES

Due to the nature of the menu processor, its routines depend on data stored in common blocks. Because the applications program may access the processor at any point in the application, the common blocks are necessary to store the menu data, pointer data, and current status data. In this way, the applications program does not interfere with the menu processing. Care must be taken by the applications programmer, however, in the selection of names for applications common blocks and subroutines. Those reserved for the menu processor are listed in this appendix.

BLOCK NAME	CONTENT DESCRIPTION
MENUS	Menu data and pointer arrays
TREE	Reject buffer and triggering pointer
APPLIC	Application file name and delimiter
GLOBAL	Global menu information
INOUT	Input and output registers
MESSAGE	Message information
TAHEAD	Type ahead buffer and pointers

Table A.1. Menu processor common blocks

DSPLY_MENU

This routine displays the current menu to the output device.

Format CALL DSPLY_MENU (*nm, itype*)

Arguments *nm, itype*

format: integer*2

access: read only

Devices Dialogue device only

Description The current menu is displayed with either key characters or option numbers to indicate valid selections

Input:

nm = the current menu number;

itype = 1 to display key characters;

= 2 to display option numbers.

GLOBAL

This routine drives the appropriate global menu.

Format CALL GLOBAL (*menu, state, io, ix, isw*)

Arguments *menu, isw*

format: integer*2

access: read only

state

format: character*4

access: read only

io, ix

format: integer*2

access: write only

Devices All

Description The global menu may be displayed according to the current menu. It may also be checked for valid options depending on the current menu and the built-in global menu.

Input:

menu = current menu number;

state = option selected by user;

isw = 1 for display purposes;

= 2 for option checking purposes.

Output:

io = pointer to *mendat* array for menu data;

ix = pointer to *key_chars* array for option data.

GLVAL

This routine determines the status of the application dependent global menu.

Format CALL GLVAL (*menu, bits*)

Arguments *menu*

format: integer*2

access: read only

bits

format: byte (32)

access: write only

Devices All

Description Each variable in the *bit* array corresponds to an application dependent global menu option. This routine translates a single integer associated with the current menu into on and off switches for the global menu.

Input:

menu = sequential menu number.

Output:

bits(i) = 1 if the *i*th option is on;

= 0 if the *i*th option is off.

INFORM

This routine writes a message to the output device.

Format CALL INFORM (*num*)

Arguments *num*

format: integer*2

access: read only

Devices All

Description A message from the menu tree structure data file is written on the output device. This routine is application program callable.

Input:

num = message number in database.

INIT_MENU

This routine initializes the menuing environment.

Format CALL INIT_MENU (*file, del*)

Arguments *file*

format: character*30

access: read and write

del

format: character*1

access: read only

Devices All

Description All variables are initialized.

Input:

file = name of tree structure data file;

del = option delimiter symbol;

LONG

This routine determines the filled portion of a string variable.

Format CALL LONG (*word, inuse, size*)

Arguments *word*

format: character**size*

access: read only

inuse, size

format: integer*2

access: read and write

Devices All

Description The used portion of a string variable is determined for output purposes.

Input:

word = string variable;

size = length of string variable;

Output:

inuse = used portion of string variable.

MENUS

This routine initializes the menu variables and drives the menu processor.

Format CALL MENUS (*file, delim, isw*)

Arguments *file*

format: character*30

access: read and write

delim

format: character*1

access: read and write

isw

format: integer*2

access: read only

Devices All

Description After the variables are initialized, READ_MENUS is called to read in data. Tree is called in such a manner to drive the menu processor.

Input:

file = name of tree structure data file;

del = delimiter character;

isw = 1 to display key characters;

= 2 to display option numbers.

READ_MENUS

This routine reads the menu tree structure data file.

Format	CALL READ_MENUS ()
Arguments	*none*
Devices	All
Description	The menu hierarchy data file is read. All of the menuing arrays are filled and ready for processing.

TREE

This routine is the main menu handling routine for the menu processor.

Format CALL TREE (*start, state, next, isw*)

Arguments *start*

format: character*30

access: read only

state

format: character*4

access: write only

next

format: character*30

access: write only

isw

format: integer*2

access: read only

Devices All

Description This routine displays the current menu and allows the user to select an option. The type ahead buffer is filled and the option is checked against the valid options for that menu. The processor updates the current menu, activates the global menu, or calls a

subroutine according the the user option and the menu hierarchy.

Input:

start = current menu name;

isw = 1 to display key characters;

= 2 to display option numbers.

Output:

state = option selected;

next = next menu to be displayed.

MENNUM

This routine determines the name or number of a menu.

Format CALL MENNUM (*name, num, index*)

Arguments *name*

format: character*30

access: read and write

num, index

format: integer*2

access: read and write

Devices All

Description If the name of a menu is specified, the corresponding menu number is returned. If the number of the menu is specified, the name is returned.

Input:

index = 0 if name is input;

= 1 if number in input;

Input/Output:

name = menu name;

num = menu number.

Appendix B

UIT$CREATE ROUTINES

Due to the interaction between the general purpose menu processor and the UIT$CREATE routines, its routines depend on data stored in common blocks. Because the information that UIT$CREATE generates will become menuing information, the common blocks and arrays have much the same format as the menu processor routines.

BLOCK NAME	CONTENT DESCRIPTION
USER	Menu data and pointer arrays
GLOBE	Global menu information
INOUT	Input and output registers
WORDS	Message information
CURRENT	Current menu pointers

Table B.1. UIT$CREATE common blocks

ADMESS

This routine adds a message to the data base.

Format CALL ADMESS

Arguments *none*

Description The user inputs a message which is added to the data.

CHECK

This routine checks the validity of the data.

Format CALL CHECK ( )

Arguments *none*

Description This routine checks that each menu pointed to exists. If a menu is pointed to and a corresponding data set does not exist, the menu is written to the screen.

CREATE_CONTROL

This routine creates CONTROL.FOR.

Format CALL CREATE_CONTROL ( )

Arguments *none*

Description The triggering subroutine CONTROL.FOR is created according to the data.

LHIER

This routine lists the menu data hierarchy.

Format CALL LHIER ( )

Arguments *none*

Description The menu hierarchy is listed to the screen from a user inputed starting point.

DELMEN

This routine deletes a menu from the data.

Format CALL DELMEN ( )

Arguments *none*

Description A user specified menu data set is deleted from the data. The common blocks are updated to reflect this change.

DELOPT

This routine deletes an option from a menu.

Format CALL DELOPT ( )

Arguments *none*

Description A user specified option is deleted from the current menu. All common blocks are updated to reflect the change.

DEMESS

This routine deletes a message from the data.

Format CALL DEMESS ( )

Arguments *none*

Description A user specified message is deleted from the message data set. The MESSIJ common block elements are updated to reflect this.

DETGLOB

This routine determines which menu is the global menu.

Format CALL DETGLOB ( )

Arguments *none*

Description This routine searches through the menu titles to find the global menu number.

LIMESS

This routine lists the messages to the screen.

Format CALL LIMESS ( )

Arguments *none*

Description This routine lists the message data set which the user has entered.

FINDNO

This routine determines the name or number of a menu.

Format CALL FINDNO (*name, num, index*)

Arguments *name*

format: character*30

access: read and write

num, index

format: integer*2

access: read and write

Description If the name of a menu is specified, the corresponding menu number is returned. If the number of the menu is specified, the name is returned. This executable lines of this routine are identical to those of MENNUM in the menuing trigger function library, however the COMMON blocks reflect the information the user has input.

Input:

index = 0 if name is input;

= 1 if number in input;

Input/Output:

name = menu name;

num = menu number.

LIST

This routine lists the current menu to the screen.

Format CALL LIST ( )

Arguments *none*

Description The menu on which the user is working is displayed to the screen. This routine is called from several different points in the menu tree.

LISTMEN

This routine lists the menu titles.

Format CALL LISTMEN ( )

Arguments *none*

Description All of the titles of the user inputed menu data sets are listed to the screen.

NEWTITLE

This routine determines the title of a new menu data set.

Format CALL NEWTITLE ( )

Arguments *none*

Description For a new menu data set, the title is specified. If the menu is to be the top level menu, the user is notified. All of the global options are activated.

OLDFIL

This routine reads in an existing data file for modifications.

Format CALL OLDFILE ( )

Arguments *none*

Description All of the data of a specified menu tree structure data file is read into the proper arrays.

ONEOH

This routine determines the global option switches of a menu.

Format CALL ONEOH (*menu, bits*)

Arguments *menu*

format: integer*2

access: read

bits

format: integer*2

access: write

Description The global menu integer of a particular menu is decoded into its global option switches.

Input:

menu = menu number;

Output:

bits = array containing 1's and 0's for switches.

OPTION

This routine adds an option to the current menu.

Format CALL OPTION ( )

Arguments *none*

Description An option is added to the current menu at the bottom of the menu data set. All of the arrays for this menu are updated to reflect the new option.

PUTBITS

This routine translates a binary number to its integer equivalent.

Format CALL PUTBITS (*menu, bits*)

Arguments *menu, bits*

format: integer*2

access: read only

Description The global option switches for a particular menu are encoded into a single integer value for easy storage.

Input:

menu = menu number;

bits = array of 1's and 0's.

SAVE

This routine saves all of the user input data to a menu tree structure data file.

Format CALL SAVE ( )

Arguments *none*

Description All of the data stored in the common blocks is written to a data file according to specification.

SAVEMENU

This routine saves menu data to common blocks.

Format CALL SAVEMENU ( )

Arguments *none*

Description All of the options of a menu are saved to the common blocks. Some of the data pointers are updated to reflect this save.

SEGLOB

This routine sets a particular menu to be the global menu.

Format CALL SEGLOB ( )

Arguments *none*

Description This routine determines which menu the user prefers to be the global menu.

SETGLO

This routine sets the current menu at the global menu.

Format CALL SETGLO ( )

Arguments *none*

Description This routine sets the current menu that the user is working with to be the global menu. If no global menu exists, the user is warned.

SPECFILE

This routine determines an input file name.

Format CALL SPECFILE (*name*)

Arguments *name*

format: character*30

access: read only

Description This routine requests the file name of a data file that is to be read for modifications.

Input:

name = data file name.

STAGLO

This routine sets the status of the global menu switches for a menu.

Format CALL STAGLO ( )

Arguments *none*

Description The routine lists the available menus to the screen for user input. It then lists the current global menu switch status of the selected menu. The user may then toggle switches.

TITLE

This routine determines the title of the current menu.

Format CALL TITLE ( )

Arguments *none*

Description The title of the current menu is obtained from the user. This routine is used for new menu data sets as well as renaming existing ones.

WHICHM

This routine determines the current menu.

Format CALL WHICHM (*num*)

Arguments *num*

format: integer*2

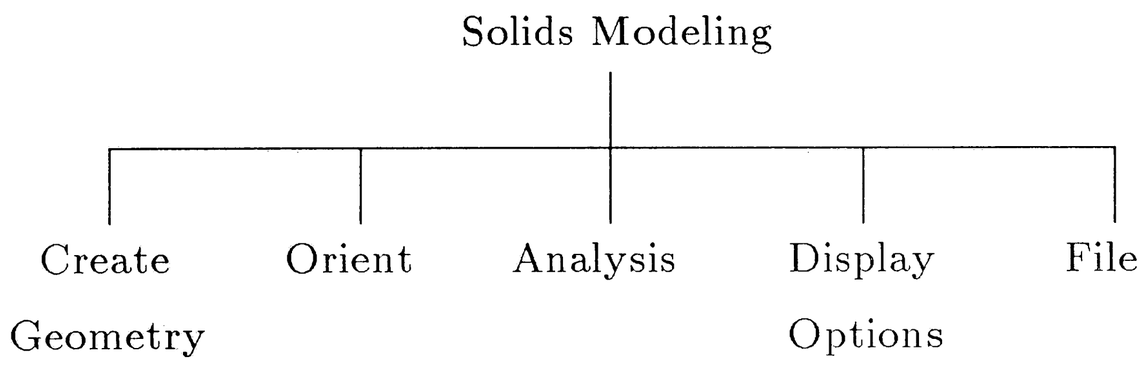
access: write

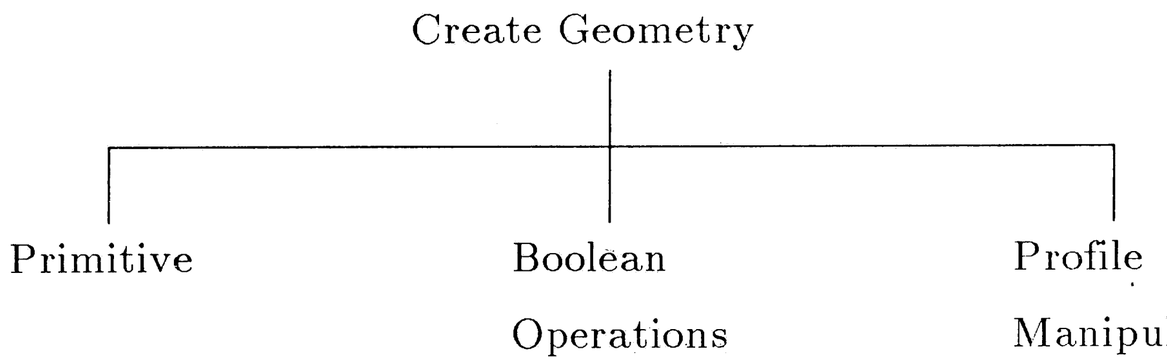
Description This routine determines the current menu for the menu modification options. All of the menu pointer information is updated to reflect the current menu.

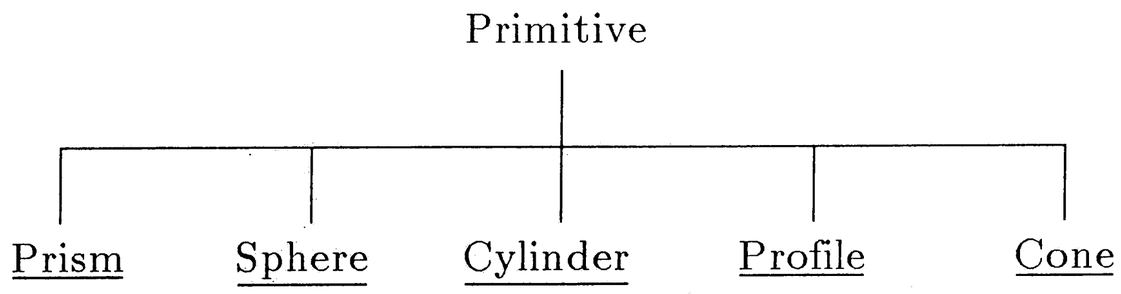
Appendix C

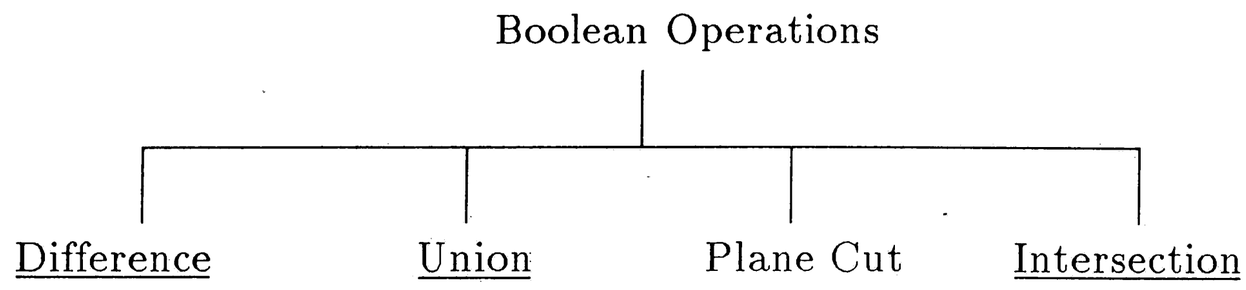
SAMPLE TREE STRUCTURE DATA FILE

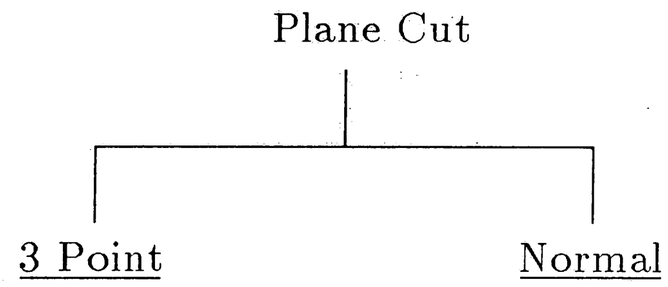
This appendix lists a sample data file of a solids modeling application whose menu tree structure follows. In this diagram, an underline denotes the end of a branch. An option with no underline points to a submenu.

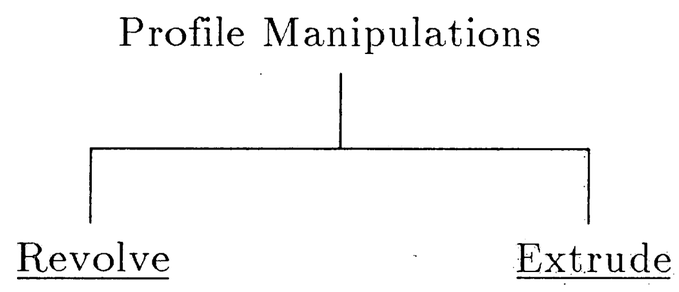


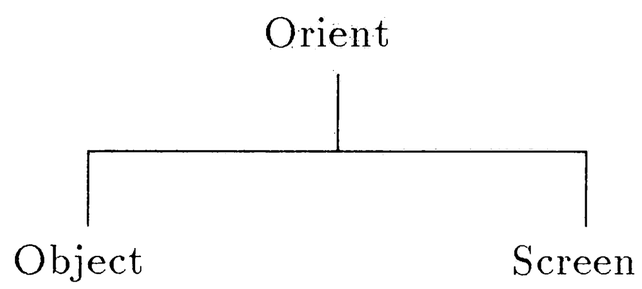


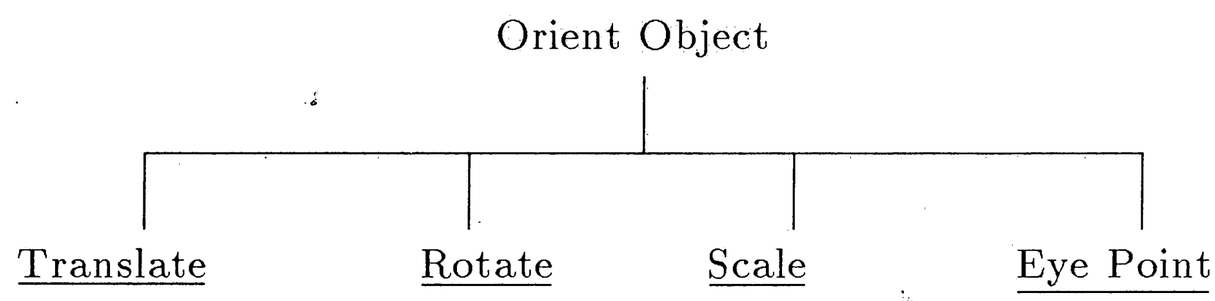


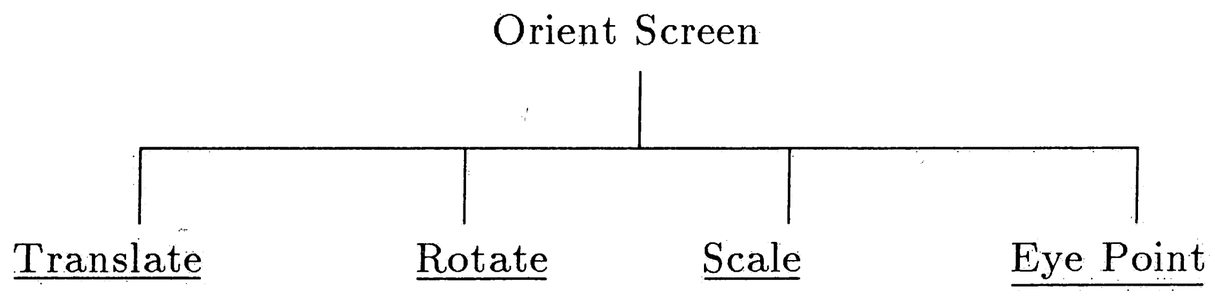

Boolean Operations
Difference
Union
Plane Cut
Intersection

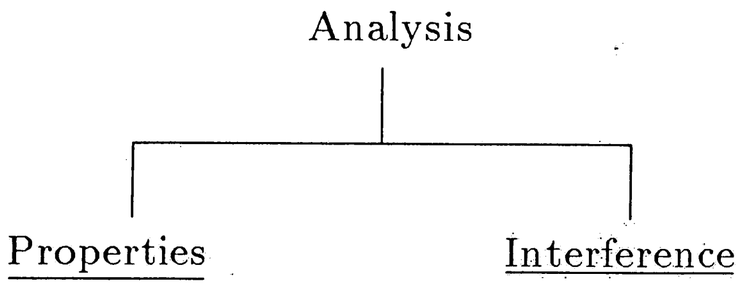

Plane Cut
3 Point
Normal

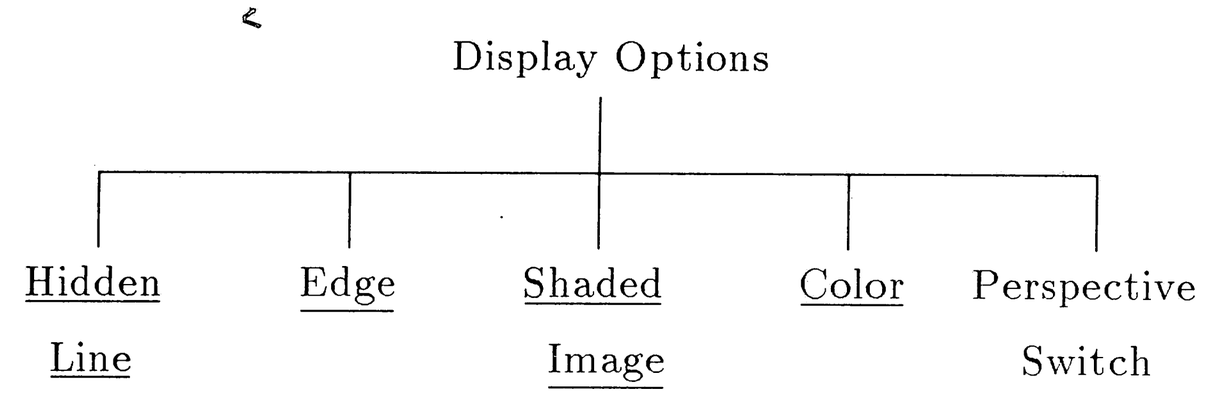

Profile Manipulations
Revolve
Extrude

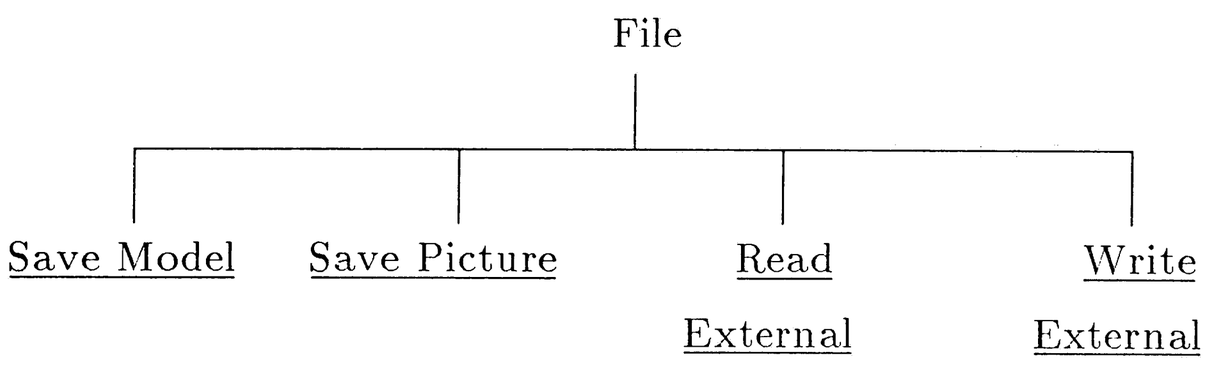

Orient
Object
Screen

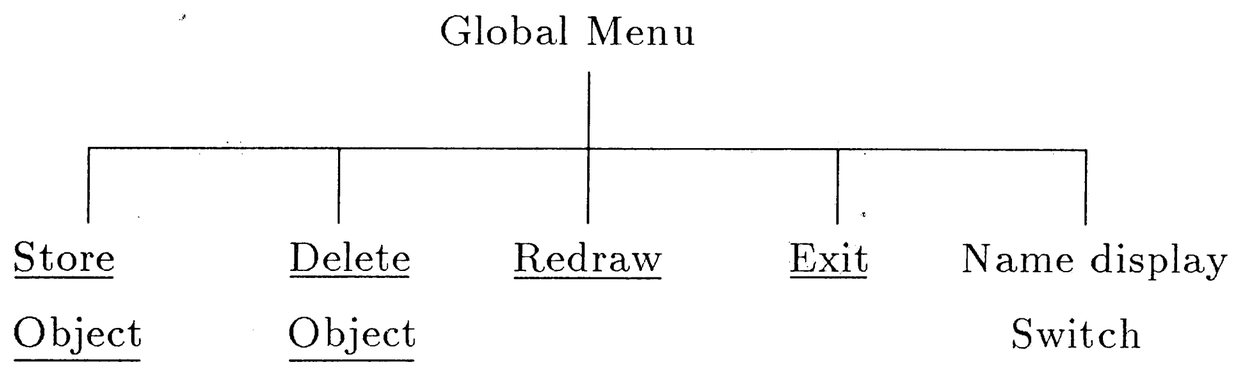

Orient Object
Translate
Rotate
Scale
Eye Point

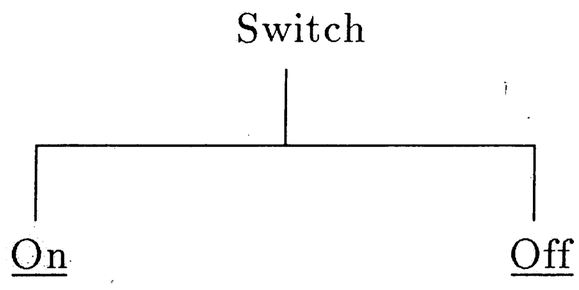

Orient Screen
Translate
Rotate
Scale
Eye Point


Analysis
Properties
Interference


Display Options
Hidden Line
Edge
Shaded Image
Color
Perspective Switch


File
Save Model
Save Picture
Read External
Write External


Global Menu
Store Object
Delete Object
Redraw
Exit
Name display Switch


Switch
On
Off

```
100
Solids Modeling                          11111
CG- Create Geometry                      1Create Geometry
OR- Orient                               1Orient
AN- Analysis                             1Analysis
DO- Display Options                      1Display Options
F- File                                  1File
-1
100
Create Geometry                          11111
PR-Primitive                             1Primitive
B- Boolean Operations                    1Boolean Operations
PM- Profile Manipulations                1Profile Manipulations
-1
100
Primitive                                11111
PRI- Prism                               2PRISM
SP- Sphere                               2SPHERE
CY- Cylinder                             2CYLNDR
PF- Profile                              2PROFIL
CO- Cone                                 2CONE
-1
100
Boolean Operations                       11111
D- Difference                            2DIFF
U- Union                                 2UNION
P- Plane Cut                             1Plane Cut
I- Intersection                          2INTSCT
-1
100
Plane Cut                                11111
3P-3 Point Definition                    2POINT3
N-Normal and Point                       2NORMAL
-1
100
Profile Manipulations                    11111
RE- Revolve                              2REVOLV
E- Extrude                               2EXTRUD
-1
100
Orient                                   11111
OB- Object                               1Orient Object
SC- Screen                               1Orient Screen
-1
100
Orient Object                            11111
TR- Translate                            2OTRANS
RO- Rotate                               2OROT
SC- Scale                                2OSCAL
EYE - Eye Point                          2OEYE
-1
```

```
100
Orient Screen                          11111
TR- Translate                          2STRANS
RO- Rotate                             2SROT
SC- Scale                              2SSCAL
EYE - Eye Point                        2SEYE
-1
100
Analysis                               11111
PR- Properties                         2PROP
I- Interference                        2INTFER
-1
100
Display Options                        11111
HL- Hidden Line                        2HIDDEN
ED- Edge Display                       2EDGE
SI- Shaded Image                       2SHADE
C- Color                               2COLOR
P- Perspective Switch                  1On / Off
-1
100
File                                   11111
SM- Save Model                         2SAVMOD
SP- Save Picture to File               2SAVPIC
RE- Read External File                 2READEX
WR- Write External File                2WRITEX
-1
100
Global Menu
ST- Store Object                       2STORE
DE- Delete Object                      2DELET
RED- Redraw                            2REDRAW
EX- Exit                               2EXIT
NA- Name Display Switch                1On / Off
-1
100
On / Off                               00000
ON- Turn switch on                     2ON
OFF- Turn switch off                   2OFF
-1
200
Enter object name:
Enter translation (x,y,z):
Enter rotation (x,y,z):
Enter scale:
Enter file name:
Enter point for primitive:
Enter cutting object name:
-1
```

Appendix D

SAMPLE UIT$CREATE SESSION

This appendix lists a sample interactive session of the UIT$CREATE program. This session begins creating the data file of appendix C. The user input is italicized.

```
$ RUN UIT$CREATE

File Access
RE- Read in existing file
NEW- Create new file

Enter Command:  NEW

Option selected: NEW- Create new file

Menu Database
CR- Create new menu branch
CH- Change menu branch
GL- Global menu status
ME- Messages
LH- List hierarchy
SA- Save entire menu tree
EX- Exit

Enter Command: CR

Option selected: CR- Create new menu branch

***** This must be your top level menu!! *****
Enter menu name (exact caps please): Solids Modeling

Create Menu Branch
AD- Add option to branch
LI- List current menu

Enter command: AD
Option selected: AD- Add option to branch

Enter option no.  1  (Include delimiter):  CG- Create Geometry
```

CG- Create Geometry points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *1*
Enter menu name: *Create Geometry*

Create Menu Branch
AD- Add option to branch
LI- List current menu

Enter command: *AD*
Option selected: AD- Add option to branch

Enter option no. 2 (Include delimiter): *OR- Orient*

OR- Orient points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *1*
Enter menu name: *Orient*

Create Menu Branch
AD- Add option to branch
LI- List current menu

Enter command: *AD*
Option selected: AD- Add option to branch

Enter option no. 3 (Include delimiter): *AN- Analysis*

AN- Analysis points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *1*
Enter menu name: *Analysis*

Create Menu Branch
AD- Add option to branch
LI- List current menu

Enter command: *AD*
Option selected: AD- Add option to branch

Enter option no. 4 (Include delimiter): *DO- Display Options*

DO- Display Options points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code

Enter choice: *1*
Enter menu name: *Display Options*

Create Menu Branch
AD- Add option to branch
LI- List current menu

Enter command: *AD*
Option selected: AD- Add option to branch

Enter option no. 5 (Include delimiter): *F- File*

File points to:
1 - Another menu
2 - Subroutine
0 - Taken care of by application code
Enter choice: *1*
Enter menu name: *File*

Create Menu Branch
AD- Add option to branch
LI- List current menu

Enter command: *LI*

Option selected: LI- List current menu

Solids Modeling
CG- Create Geometry points to Create Geometry
OR- Orient points to Orient
AN- Analysis points to Analysis
DO- Display Options points to Display Options
F- File points to File
Create Menu Branch
D- Add option to branch
I- List current menu

Enter Command: \

Menu Database
CR- Create new menu branch
CH- Change menu branch
GL- Global menu status
ME- Messages
LH- List hierarchy
SA- Save entire menu tree
EX- Exit

Enter Command: * *CR*

Option selected: Toggle menu display off

Enter menu name (exact caps please): *Create Geometry*

Enter command: */ CR*

Option Selected: CR- Create new menu branch

Enter menu name (exact caps please): *Primitive*

Enter command: */ CR*

Enter menu name (exact caps please): *Boolean*

Enter command: */ CR*

Enter menu name (exact caps please): *Plane Cut*

Enter command: *AD*

Enter option no. 1 (Include delimiter): *3P- 3 Point Definition*

3P- 3 Point Definition points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *POINT3*

Enter command: *AD*

Enter option no. 2 (Include delimiter): *N- Normal and Point*

N- Normal and Point points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *NORMAL*

Enter command: */ CR*

Enter menu name (exact caps please): *Profile Manipulations*

Enter command: */ CR*

Enter menu name (exact caps please): *Orient*

Enter command: */ CR*

Enter menu name (exact caps please): *Orient Object*

Enter command: */ CR*

Enter menu name (exact caps please): *Orient Screen*

Enter command: */ CR*

Enter menu name (exact caps please): *Analysis*

Enter command: *! CR*

Enter menu name (exact caps please): *Display Options*

Enter command: */ CR*

Enter menu name (exact caps please): *Global Menu*

Enter command: *AD*

Enter option no. 1 (Include delimiter): *ST- Store Object*

ST- Store Object points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *STORE*

Enter command: *AD*

Enter option no. 2 (Include delimiter): *DE- Delete Object*

DE- Delete Object points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *DELET*

Enter command: *AD*

Enter option no. 3 (Include delimiter): *RED- Redraw*

RED- Redraw points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *REDRAW*

Enter command: *AD*

Enter option no. 4 (Include delimiter): *EX- Exit*

EX- Exit points to:

1 - Another menu
2 - Subroutine
0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *EXIT*

Enter command: *AD*

Enter option no. 5 (Include delimiter): *NA- Name Display Switch*

NA- Name Display Switch points to:
1 - Another menu
2 - Subroutine
0 - Taken care of by application code
Enter choice: *1*
Enter subroutine name: *On / Off*

Enter command: */ CR*

Enter menu name (exact caps please): *On / Off*

Enter command: */ **

Option selected: Toggle menu display on

Menu Database
CR- Create new menu branch
CH- Change menu branch
GL- Global menu status
ME- Messages
LH- List hierarchy
SA- Save entire menu tree
EX- Exit

Enter command: CH

Option Selected: CH- Change menu branch

Change Menu
CT- Change title of menubranch
DB- Delete menu branch
LM- List menu names
CO- Change option on menu

Enter command: *CT*
Option Selected: CT- Change Title of menubranch

Select Menu by Number:
1 Solids Modeling
2 Create Geometry
3 Primitive
4 Boolean

5 Plane Cut
6 Profile Manipulations
7 Orient
8 Orient Object
9 Orient Screen
10 Analysis
11 Display Options
12 File
13 Global Menu
14 On / Off
999 No selection

Enter number: *4*
Enter menu name (exact caps please): *Boolean Operations*

Change Menu
CT- Change title of menubranch
DB- Delete menu branch
LM- List menu names
CO- Change option on menu

Enter command: *Co*

Option Selected: CO- Change option on menu

Select Menu by Number:
1 Solids Modeling
2 Create Geometry
3 Primitive
4 Boolean Operations
5 Plane Cut
6 Profile Manipulations
7 Orient
8 Orient Object
9 Orient Screen
10 Analysis
11 Display Options
12 File
13 Global Menu
14 On / Off
999 No selection

Enter number: *3*

Change Option
AD- Add option to branch
DE- Delete option from branch
RE- Replace option on branch
LI- List current menu branch

Enter command: *AD*

Option selected: AD- Add option to branch

Enter option no. 1 (Include delimiter): *PRI- Prism*

PRI- Prism points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *PRISM*

Change Option
AD- Add option to branch
DE- Delete option from branch
RE- Replace option on branch
LI- List current menu branch

Enter command: *AD*

Enter option no. 2 (Include delimiter): *SP- Sphere*

N- Normal and Point points to:
    1 - Another menu
    2 - Subroutine
    0 - Taken care of by application code
Enter choice: *2*
Enter subroutine name: *SPHERE*

Change Option
AD- Add option to branch
DE- Delete option from branch
RE- Replace option on branch
LI- List current menu branch

Enter command: *!!*

Menu Database
CR- Create new menu branch
CH- Change menu branch
GL- Global menu status
ME- Messages
LH- List hierarchy
SA- Save entire menu tree
EX- Exit

Enter Command: *ME*

Option selected: ME- Messages

Messages
AM- Add message to list
DM- Delete message from list

RE- Replace message in list
LM- List messages

Enter Command: *AM*

Option selected: AM- Add message to list

Enter message no. 1: *Enter object name:*

Messages
AM- Add message to list
DM- Delete message from list
RE- Replace message in list
LM- List messages

Enter command: *

Option selected: Turn off menu display

Enter command: *AM*

Enter message no. 3: *Enter translation (x,y,z):*

Enter command: *AM*

Enter message no. 3: *Enter rotation (x,y,z):*

Enter command: *AM*

Enter message no. 4: *Enter scale:*

Enter command: *AM*
Enter message no. 5: *Enter file name:*

Enter command: *AM*

Enter message no. 6: *Enter point for primitive:*

Enter command: *AM*

Enter message no. 7: *Enter cutting object name:*

Enter command: / GL *

Global Menu Status
AD- Add option to global menu
ST- Set status of global menu
RE- Select menu to be global
LI- List global menu

Enter command: *RR*

Invalid command RR

Enter command: *ST*

Option selected: ST- Set status of global menu

Select Menu by Number:
1 Solids Modeling
2 Create Geometry
3 Primitive
4 Boolean
5 Plane Cut
6 Profile Manipulations
7 Orient
8 Orient Object
9 Orient Screen
10 Analysis
11 Display Options
12 File
13 Global Menu
14 On / Off
999 No selection
Enter number: *14*

Status for On / Off

1 ST- Store Object	ON
2 DE- Delete Object	ON
3 RED- Redraw	ON
4 EX- Exit	ON
5 NA- Name Display Switch	ON

999 No selection

Enter switch to toggle: *1*

Status for On / Off

1 ST- Store Object	OFF
2 DE- Delete Object	ON
3 RED- Redraw	ON
4 EX- Exit	ON
5 NA- Name Display Switch	ON

Global Menu Status
AD- Add option to global menu
ST- Set status of global menu
RE- Select menu to be global
LI- List global menu

Enter command: * \ SA *

Option selected: Toggle display switch off

Option selected: Toggle display switch on

Save Tree
CHK- Check tree
SA- Save tree to file

Enter command: *CHK*

The following menus do not exist:
File
End of menu list

Save Tree
CHK- Check tree
SA- Save tree to file

Enter command: / *CR*

Enter menu title (exact caps please): File

Enter command: / *SA SA*

Enter save file name: *SOLIDS.DAT*

Enter command: EX

$

VITA

Catherine Mary Curtin was born on January 3, 1962, in Hartford, Connecticut. She is the daughter of Charles Miller Curtin and Margaret Simon Curtin.

She was graduated from Lehigh University with a Bachelor of Science in Mechanical Engineering in June of 1984. After graduation, she joined Digital Equipment Corporation as a software engineer.

On February 22, 1986, she married Joseph Thomas Clifford. Soon after, they moved to Groβ Karben, West Germany. Both Mr. Clifford and Ms. Curtin returned to graduate school at Lehigh in the fall of 1987. Ms. Curtin was a teaching assistant in the College of Business and Economics during her graduate education.

As of this writing, Ms. Curtin is looking forward to a long summer vacation before she continues with her engineering career.