BACKGROUND MONITOR FOR THE SATCOM SIGNAL ANALYZER

Heinz-Joachim Niemann

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

BACKGROUND MONITOR FOR THE SATCOM SIGNAL ANALYZER

by

Heinz-Joachim Niemann

December 1979

Thesis Advisor: J. E. Ohlson

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BACKGROUND MONITOR FOR THE SATCOM SIGNAL ANALYZER		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1979
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Heinz-Joachim Niemann		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1979
		13. NUMBER OF PAGES 200
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Satellite Signal Analyzer
Operator Interaction

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A SATCOM Signal Analyzer (SSA) is being developed by the Satellite Communications Laboratory of the Naval Postgraduate School. The purpose of this system is to provide high-speed multi-channel digital spectrum analysis and characterization of the outputs of UHF communication satellite transponders while in orbit and operating. A Digital Computer Subsystem provides the control for most of the equipment in the system.

The thesis documents a portion of the software development for this subsystem. This software performs all operator interaction using a Touch Input System and a Graphics Display. It is controlled to a large extent by menu data stored in a disc file, which can easily be extended or modified.

Non-trivial file management and concurrent processing of several functions are major characteristics of this software.

BACKGROUND MONITOR FOR THE SATCOM SIGNAL ANALYZER

by

Heinz-Joachim Niemann
Kapitaenleutnant, Federal German Navy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1979

ABSTRACT

A SATCOM Signal Analyzer (SSA) is being developed by the Satellite Communications Laboratory of the Naval Postgraduate School. The purpose of this system is to provide high-speed multi-channel digital spectrum analysis and characterization of the outputs of UHF communication satellite transponders while in orbit and operating. A Digital Computer Subsystem provides the control for most of the equipment in the system.

The thesis documents a portion of the software development for this subsystem. This software performs all operator interaction using a Touch Input System and a Graphics Display. It is controlled to a large extent by menu data stored in a disc file, which can easily be extended or modified.

Non-trivial file management and concurrent processing of several functions are major characteristics of this software.

TABLE OF CONTENTS

I. INTRODUCTION - 11

A. BACKGROUND - - - - - - - - - - - - - - - - - - - 11

B. NATURE OF THE PROBLEM - - - - - - - - - - - - - 13

C. SCOPE OF THIS REPORT - - - - - - - - - - - - - - 13

II. SSA DIGITAL COMPUTER SUBSYSTEM - - - - - - - - - - 14

A. INTRODUCTION - - - - - - - - - - - - - - - - - - 14

B. SOFTWARE REQUIREMENTS - - - - - - - - - - - - - 14

III. SSA MONITOR SOFTWARE - - - - - - - - - - - - - - - 16

A. INTRODUCTION - - - - - - - - - - - - - - - - - - 16

B. OPERATING SYSTEMS - - - - - - - - - - - - - - - 16

C. RATFOR LANGUAGE - - - - - - - - - - - - - - - - 17

D. SYSTEM SOFTWARE - - - - - - - - - - - - - - - - 18

1. Foreground Monitor - - - - - - - - - - - - - 18

2. Background Monitor - - - - - - - - - - - - - 18

3. Foreground/Background Communication - - - 18

E. SUPPORT SOFTWARE - - - - - - - - - - - - - - - - 19

1. Introduction - - - - - - - - - - - - - - - - 19

2. Menu Editor - - - - - - - - - - - - - - - - 19

3. Menu Checker - - - - - - - - - - - - - - - - 19

IV. BACKGROUND MONITOR - - - - - - - - - - - - - - - - 20

A. INTRODUCTION - - - - - - - - - - - - - - - - - - 20

B. HARDWARE ENVIRONMENT - - - - - - - - - - - - - - 20

1. Introduction - - - - - - - - - - - - - - - - 20

2. Hewlett-Packard 2649A Terminal - - - - - - 20

3. Carroll Touch Input System - - - - - - - - 21

C. SOFTWARE STRUCTURE - - - - - - - - - - - - - - - 22

1. Introduction- - - - - - - - - - - - - - - - - - 22

2. Background Controller - - - - - - - - - - - - - 23

3. Background Tasks- - - - - - - - - - - - - - - - 23

V. BACKGROUND CONTROLLER - - - - - - - - - - - - - - - - 24

A. INTRODUCTION- 24

B. COMMON DATA - 28

1. Menu Table- - - - - - - - - - - - - - - - - - - 28

2. Menu Stack- - - - - - - - - - - - - - - - - - - 28

3. Message Stack - - - - - - - - - - - - - - - - - 29

C. EXECUTIVE - 29

1. Common Routine ´endfbk´ - - - - - - - - - - - - 30

2. Common Routine ´enhanc´ - - - - - - - - - - - - 30

3. Common Routine ´errmsg´ - - - - - - - - - - - - 31

4. Common Routine ´ascint´ - - - - - - - - - - - - 31

D. CONTROL I/O MANAGER - - - - - - - - - - - - - - - - 32

E. INPUT MANAGER - - - - - - - - - - - - - - - - - - - 32

F. CONTROL COMMAND MANAGER - - - - - - - - - - - - - - 33

G. MENU MANAGER- 34

1. Menu Display- - - - - - - - - - - - - - - - - - 34

2. Menu Stack ´menstk´ - - - - - - - - - - - - - - 36

3. Menu Hierarchy- - - - - - - - - - - - - - - - - 36

4. Menu Structure- - - - - - - - - - - - - - - - - 37

5. Functions - - - - - - - - - - - - - - - - - - - 38

H. PARAMETER BLOCK MANAGER - - - - - - - - - - - - - - 40

I. LOG MANAGER - 40

K. MESSAGE MANAGER - - - - - - - - - - - - - - - - - - 40

1. Input Related Messages- - - - - - - - - - - - - 41

2. Warning Messages- - - - - - - - - - - - - - - - 42

VI. BACKGROUND TASKS- 45

A. INTRODUCTION- 45

B. BACKGROUND TASK STRUCTURE - - - - - - - - - - - - 45

VII. SUPPORT SOFTWARE- 47

A. INTRODUCTION- 47

B. MENU EDITOR - 47

1. Introduction- - - - - - - - - - - - - - - - - - 47

2. Files - 48

3. Functions - - - - - - - - - - - - - - - - - - - 50

a. Initial Start Menu- - - - - - - - - - - 50

b. Create a Menu - - - - - - - - - - - - - - 50

c. Update a Menu - - - - - - - - - - - - - - 51

d. Delete a Menu - - - - - - - - - - - - - - 51

C. MENU CHECKER- 51

1. Introduction- - - - - - - - - - - - - - - - - - 51

2. Functions - - - - - - - - - - - - - - - - - - - 51

VIII. BACKGROUND MONITOR FILES- - - - - - - - - - - - - - - 53

A. INTRODUCTION- 53

B. MENU FILE - 53

C. MENU STACK FILE - - - - - - - - - - - - - - - - - - 57

D. MESSAGE STACK FILE- - - - - - - - - - - - - - - - 57

E. LOG FILE- 57

IX. CONCLUSIONS - 58

APPENDIX A SUMMARY OF SOFTWARE AND DOCUMENTATION - - - 60

APPENDIX B MENU EDITOR MANUAL- - - - - - - - - - - - - - 61

APPENDIX C MENU CHECKER MANUAL - - - - - - - - - - - - - 72

SAMPLE OUTPUT OF THE MENU CHECKER- - - - - - - - - - - - - - 74

BACKGROUND CONTROLLER PROGRAM- - - - - - - - - - - - - - - - - 79

MENU EDITOR PROGRAM- 135

MENU CHECKER PROGRAM - 181

PATCH FOR RT-11- 197

LIST OF ABBREVIATIONS- 198

LIST OF REFERENCES - 199

INITIAL DISTRIBUTION LIST- - - - - - - - - - - - - - - - - - - 200

LIST OF FIGURES

1.1 Structure of the SSA- - - - - - - - - - - - - - - - - - - 12

2.1 SSA Digital Computer Subsystem- - - - - - - - - - - - - 15

5.1 Structure of the Background Controller- - - - - - - - 26

5.2 HP Terminal Display Layout- - - - - - - - - - - - - - - 35

5.3 Structure of the Menu Manager - - - - - - - - - - - - - 39

5.4 Message Stack Illustration- - - - - - - - - - - - - - - 44

8.1 Menu File Layout- 56

ACKNOWLEDGEMENT

I would like to thank Dr. John E. Ohlson and Dr. Craig Chester for their assistance in the development of this thesis and also my wife for her understanding during this period.

I. INTRODUCTION

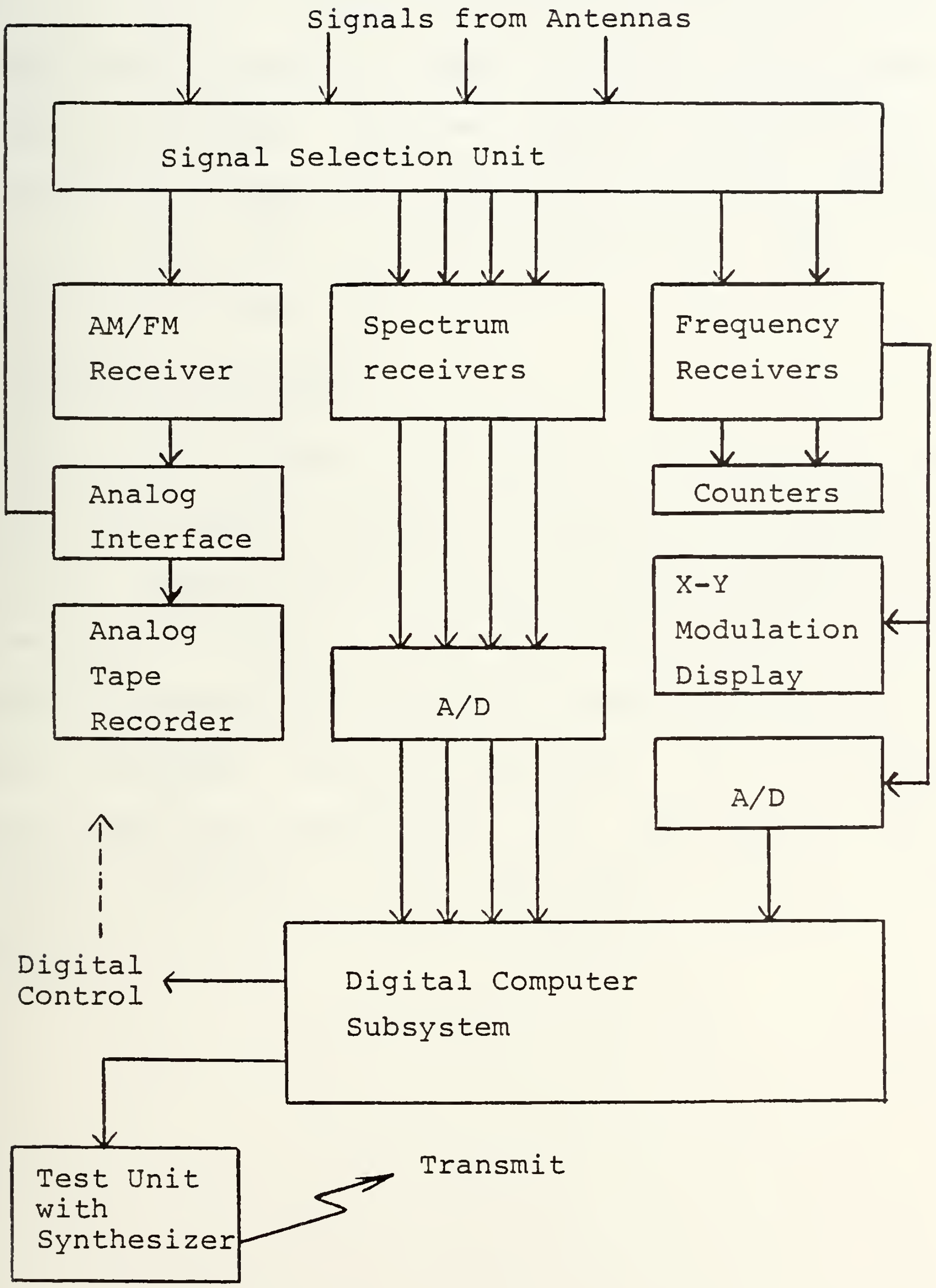
A. BACKGROUND

This project is part of the SATCOM Signal Analyzer (SSA) being developed by the Satellite Communications Laboratory of the Naval Postgraduate School. It is a prototype that will be delivered to NAVCOMSTA Stockton for operational and reliability evaluation.

The purpose of the SSA is to provide the user a means of doing spectrum monitoring continuously and to permit a method of making changes in the spectrum processing with ease. The technical characteristics of this system are (1) a minicomputer for control, (2) array processors for analysis, (3) multiple receiver channels, (4) real-time displays, and (5) hardcopy output. Fig. 1.1 gives a simplified diagram of this system.

A Digital Computer Subsystem provides the control for most of the equipment. It is constructed around a DEC PDP-11/34 minicomputer.

Figure 1.1
Structure of the SSA



B. NATURE OF THE PROBLEM

Real-time monitoring, processing of operator I/O, and control of equipment has to be done by the software of the PDP-11/34 minicomputer. Beside processing of time-critical events the operator should be given the possibility of initiating several functions which can be distinguished by priority and volume. While any number of functions are being performed it has to be possible to always initialize other functions that are more important at a given time by interrupting current ones.

C. SCOPE OF THIS REPORT

This report documents part of the software development for the SSA Digital Computer Subsystem. The documentation will give a general view of the SSA Digital Computer Subsystem and its software. It will cover in detail the Background (BG) Monitor which performs operator interaction and certain hardware control functions. Also the Support Software which was developed for the BG Monitor will be presented. Appendix A lists the present software and documentation.

II. SSA DIGITAL COMPUTER SUBSYSTEM

A. INTRODUCTION

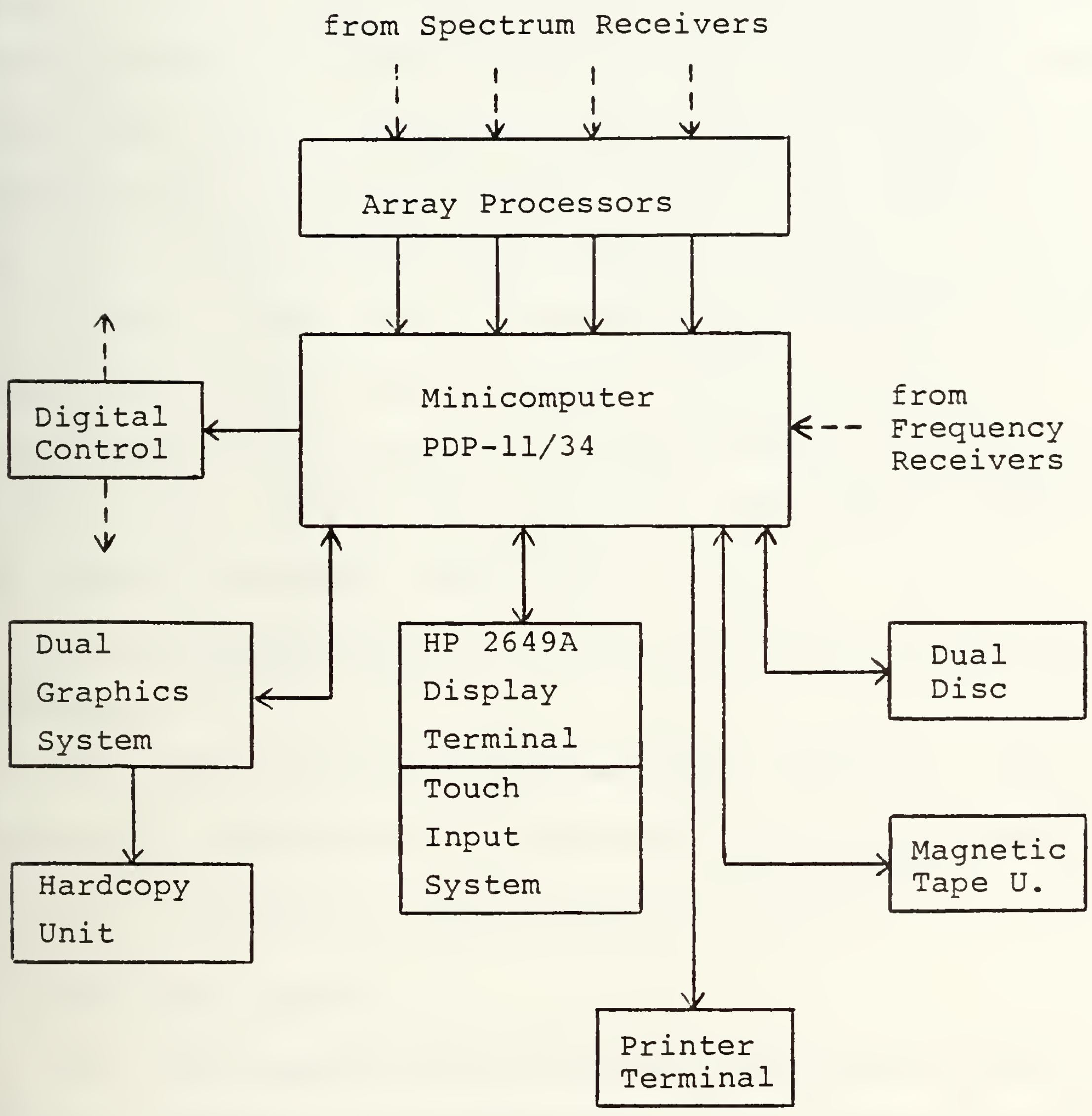
The operation of the SSA is controlled by the SSA Digital Computer Subsystem. This subsystem processes various inputs from array processors, frequency receivers, AM/FM receivers, and operator control devices. It controls synthesizers, receivers, a hardcopy unit, and monitoring devices. Fig. 2.1 illustrates the SSA Digital Computer Subsystem.

B. SOFTWARE REQUIREMENTS

The Digital Computer Subsystem performs different kinds of functions. Processing of time-critical functions has the highest priority in this system. Operator interaction is processed at the next priority level. It includes functions like change of system parameters, requests for certain displays, and hardcopy. If none of the above processing has to be done, lower priority functions will be executed. These "Background Tasks" include hardware operation tests, report generation, post analysis, etc.

Figure 2.1

SSA Digital Computer Subsystem



III. SSA MONITOR SOFTWARE

A. INTRODUCTION

The SSA Monitor software falls into two categories. The System Software will be a resident part of the SSA. The Support Software is used to create a data file which provides the System Software with necessary information to display images on the HP Terminal and process operator inputs. The System Software development has been accomplished in two steps. Most of the program editing, testing, and debugging was done using the UNIX operating system. The second step was to transfer the system software to RT-11 and perform final testing and debugging. The Support Software was entirely developed under UNIX.

The programming language RATFOR was used for software development.

This chapter gives an overview of the different software components. More detailed information about the BG Monitor software is presented in the following chapters.

B. OPERATING SYSTEMS

There were several the reasons for using two different operating systems in the development of the system software. RT-11, which was chosen for the SSA Digital Computer Subsystem, had some disadvantages compared to UNIX. The editing functions and the file management of UNIX were superior to RT-11. Also RT-11 allowed no time-sharing. In addition the UNIX RATFOR preprocessor could be used to create FORTRAN

programs. This allowed programming in a more structured way. The RATFOR programs were easier to write, read, test, and debug.

RT-11 in its Foreground/Background version is a single user, two job system restricted to 28K words of memory. In this foreground/background environment, two independent programs can reside in memory. The foreground program is given priority and executes until it relinquishes control to the background program; the background program executes until control is again required by the foreground job.

RT-11 provides the user with several programmed requests. The feature of invoking user-written completion routines was used to a large extent in the BG Monitor. A completion routine has to be scheduled using a programmed request. It then is entered in a queue and will be called by RT-11 when the specified event has happened. Such an event may be a certain time lapse or the completion of some I/O. Thus RT-11 provided five different levels of program execution:

- interrupt routine execution
- foreground completion routine execution
- foreground program execution
- background completion routine execution
- background program execution

C. RATFOR LANGUAGE

The programming language FORTRAN was chosen to be implemented in the SSA Digital Computer Subsystem running under

RT-11. As FORTRAN is an unpleasant language to some extent, the RATFOR preprocessor of the UNIX system was used to conceal the main deficiencies of FORTRAN. Thus decent control flow statements, free form input, and some other features were available. RATFOR programs were easier to write, to read, to debug, and to maintain.

D. SYSTEM SOFTWARE

1. Foreground Monitor

This program mainly is interrupt driven to control certain time critical processes, like recording data on disc and processing data from array processors. The Foreground (FG) Monitor is a resident part of the System Software. Its functions are controlled by parameters set by BG Monitor.

2. Background Monitor

The main functions of this program are processing of operator interaction and low priority BG Tasks. It executes as a background job. Part of the BG Monitor is resident, whereas BG Tasks are stored on an image file to be swapped in by RT-11 when their functions are needed. The following chapters will further document the BG Monitor.

3. Foreground/Background Communication

FG Monitor and BG Monitor communicate with each other using the communication services provided by RT-11. They allow sending data messages in either direction. The data that has to be shared by both Monitors is the Parameter Block (PB) and the Status Block (SB). The PB contains all parameters to control the functions of the FG Monitor. It

is updated by the BG Monitor and then as a whole sent to the FG Monitor. The SB provides all information about the system status. It is updated by the FG Monitor and on request transferred to the BG Monitor.

E. SUPPORT SOFTWARE

1. Introduction

The functions of the BG Monitor are to a large extent based on data stored on a menu file. These menus contain all information necessary to display a requested image at the operator control console and to process related inputs from the operator. This menu file is created, updated and checked by the Support Software which runs off-line under UNIX.

2. Menu Editor

This program updates the menu file by processing an input file that contains necessary information in standard formats. Based on general information about a menu it generates a set of display commands to create an image. Also, all necessary information to process inputs from either the keyboard or the Touch Input System is prepared and stored in the menu file.

3. Menu Checker

This program allows checking of all information that is stored on the Menu File. It extracts all requested information and presents it in a readable format. Thus any modification of the menu file can be verified.

IV. BACKGROUND MONITOR

A. INTRODUCTION

This chapter will present the Background Monitor, which executes in the background environment of RT-11. The hardware used by the Background Monitor will be addressed first. Then the general software structure and detailed software description will follow.

Special terminology used to document the Background Monitor will be introduced when appearing the first time in the text.

B. HARDWARE ENVIRONMENT

1. Introduction

The hardware used for the Background Monitor was constructed around the DEC PDP-11/34 minicomputer. Most of the equipment was standard peripheral devices. A dual disc drive was used which provided an on-line storage capacity of 80 megabits. A daisy wheel printer terminal was used to provide hardcopy output and its keyboard served as a backup for the Touch Input System. Off-line storage for transferring extracted output data and software was provided by a magnetic tape unit. A hardcopy unit was used to output exact copies of graphics displays. Finally the HP Terminal and the Touch Input System have to be addressed. Their main features are described in the following paragraphs.

2. Hewlett-Packard 2649A Terminal

This terminal has a 5 inch by 10 inch rectangular

display providing 24 lines of up to 80 characters. Some of the terminal´s capabilities are screen up and down rolling, insertion and deletion of characters and lines, cursor sensing and addressability, tabulation, margin setting, line drawing, and different character sets.

The image is self-refreshing and portions of the image can be displayed in different modes, like blinking, reverse, halfbright, or a combination thereof. All operations are controlled by the terminal´s microprocessor.

3. Carroll Touch Input System

The system provides a means for the operator to interact with the BG Monitor. The Touch Input System is implemented by utilizing a scanning infrared beam technology. A panel is attached to the Hewlett-Packard (HP) 2649A Terminal in such a way that the screen is surrounded by LED emitters on the bottom and left side and phototransistor detectors on the right side and top. Thus they form a grid of infrared light beams across the display area. To activate the input system, the operator touches the CRT screen and then the X,Y coordinates of the touch point are transmitted to the host computer.

The Touch Input System provides some benefits in contrast to a keyboard. (1) The operator does not have to alternate between looking at the display for instructions and the keyboard to respond to the system. (2) The touch targets can be located at different positions; thus position becomes a software design parameter. (3) Only the valid in-

put options at a particular situation need to be displayed, thus eliminating confusion and input errors. (4) The legend of each touch target is determined by the data displayed on the screen. It is easy to change and there is a large variety of possible legends.

C. SOFTWARE STRUCTURE

1. Introduction

There existed two constraints for the development of the BG Monitor. (1) There was a limit in available memory in the minicomputer and (2) the execution time had to be fast enough to react to operator requests in quasi real time.

For these reasons there was no display image generation done by the BG Monitor. Rather the set of display commands for each image (menu) were stored on disc in the Menu File. Additionally, all BG Tasks, which perform mutually independent functions, were linked as memory overlays on a RT-11 memory image file. They were swapped in by RT-11 when they were to be executed. Thus the structure of the BG Monitor shows a computer resident program part (BG Controller) and non-resident program parts (BG Tasks).

The RT-11-provided swapping of BG Tasks allows memory independent growth of the BG Monitor. Only the size of a BG Task is limited, not their number.

Besides this extendability, ease of maintainability was supported. As far as possible, program and data were separated. Sharing of data by the program´s modules was

kept minimal.

2. Background Controller

This resident part of the BG Monitor was designed to be highly independent of any BG Task or data on the menu file. Its components are the Executive and several Managers. The BG Controller processes requests by the operator or any of the BG Tasks.

Two different levels of program execution are implemented. Most of the BG Controller's modules run on a completion routine level (scheduled by the program, called by RT-11). The remaining parts execute as BG mainline programs.

3. Background Tasks

A BG Task performs a group of related functions. The tasks are independent of each other. Only one BG Task is executed at a time. Using the overlay feature of RT-11, these BG Tasks all reside in the same memory area.

V. BACKGROUND CONTROLLER

A. INTRODUCTION

The following paragraphs give a detailed documentation of the resident part of the BG Monitor. Additional information is embedded in the program listings.

The BG Controller has the following components:

- Executive
- Control I/O Manager
- Input Manager
- Control Command Manager
- Menu Manager
- Parameter Block Manager
- Log Manager
- Message Manager

The Executive consists of the main program and some Common Routines called by other BG Managers. Any request for operator input by the Executive or a BG Task is processed by the Control I/O Manager. Operator inputs from either keyboard or Touch Input System are processed by the Input Manager. The Control Command Manager interpretes inputs for the BG Controller. The Menu Manager updates the display image and provides the BG Controller with all information about possible inputs. Updating of the Parameter Block for the FG Monitor is done by the Parameter Block Manager. If requested the Log Manager logs all operator inputs. Finally the Message Manager displays and erases all warning messages

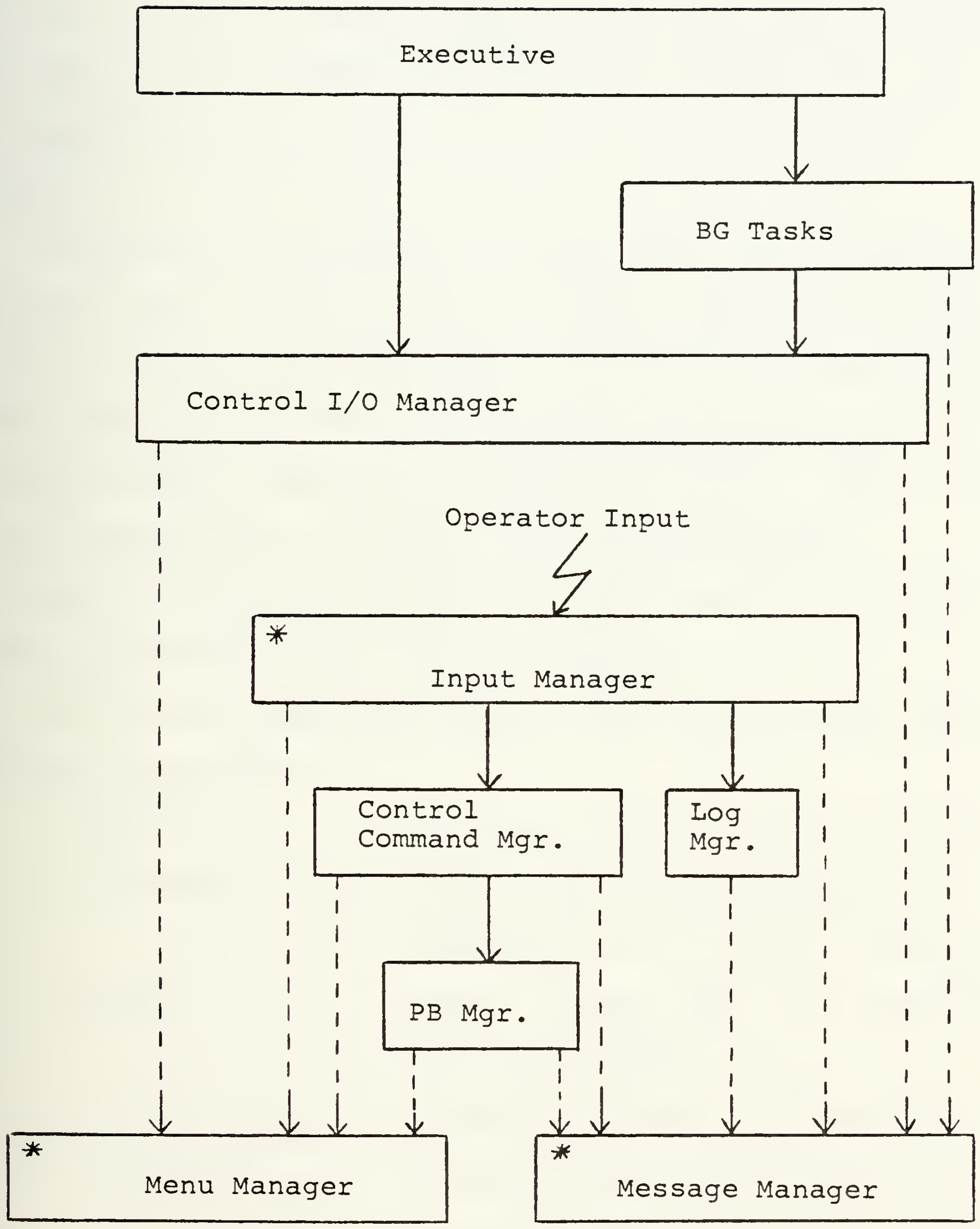
and input related messages.

Most of the BG Controller Managers execute on completion routine level. The Input Manager re-schedules itself periodically to process any operator input. Thus this Manager and all other Managers called from it (Control Command Manager, Parameter Block Manager, Log Manager) execute on completion routine level.

The Menu Manager has to perform certain functions after some time lapse (i.e. erasing of a displayed menu some time after an input occurred). On the other hand the Menu Manager may be invoked based on a BG Task request or based on an operator input. Scheduling the call takes advantage of the RT-11-provided queueing function and thus avoids any conflict.

Similar reasoning applies to the Message Manager being a completion routine. It may be invoked by all other BG Monitor programs, which either run on completion routine level or as BG mainline programs. Fig. 5.1 shows the structure of the BG Controller.

Figure 5.1
Structure of the BG Controller



Legend: * Completion Routine
—— Called as Mainline Program
- - - Called as Completion Routine

The BG Controller provides the operator with an interesting feature. It is possible to interrupt any active function of this program by selection of another function. This may create a chain of unfinished functions, and when the last one is finished the former one automatically will be invoked again.

Example:

The operator has started a BG Task that performs certain hardware tests. While this is being done he may invoke another function to have some documentation displayed. While doing this, the operator wants the system status and he starts another function to display the status. Even now he may invoke again an already interrupted function.

When the operator disables an active function, the most recent interrupted function will continue.

The following text uses certain fixed terms, whose meanings are given below:

- button:	displayed button on the screen
- permanent button:	button always being displayed on the screen at a fixed location
- menu:	display image with all possible input information
- input option:	information about a possible input
- menu-id:	identification of a certain menu; it is the number of a menu´s first data record in the menu file
- task menu:	menu which is initiated by any BG

Task

- controller menu: menu which is initiated by the resident BG Controller

B. COMMON DATA

This paragraph will document important common data which are used by more than one BG Manager.

1. Menu Table

The one-dimensional array ´mentbl´ is used to reserve memory locations to store display image and input options of a menu. The lower part of this array contains the input options of the permanent buttons. This information stays there all the time. The upper half of the array is used to temporarily store menu records with display images. After the display image has been transferred to the HP terminal, this array will store the remaining input options of a specific menu.

2. Menu Stack

The BG Controller allows certain menus to ´overwrite´ other menus being displayed. In order to get back to previously displayed menus, their menu-ids are stacked in ´menstk´. This menu stack can store up to 20 menu-ids. Should an overflow occur, the whole stack will be saved in the menu stack file. In case there is a stack underflow and a menu stack is saved in the file, it will be retrieved from the file. Thus the last initiated menu is displayed first (last in, first out).

3. <u>Message Stack</u>

The array ´msgstk´ is used as a stack for saved warning messages. It consists of an upper and a lower half. In case of a stack overflow, the lower half is saved in the message stack file to provide empty entries for any more warning messages. Then new messages always will be entered into the lower half and saved in the file if necessary.

Retrieval of warning messages from the stack is done from the upper half. If it is empty, the oldest saved stack in the menu file will be stored in the upper half of the stack.

This method assures that the oldest messages will always be displayed next (first in, first out). To save memory, the messages are stored in variable length format.

C. EXECUTIVE

The Executive of the BG Controller consists of the main program and the Common Routines. The Common Routines are used by more than one BG Manager. The Executive is called by RT-11 when the BG Controller is started and executes as a BG mainline program. The Executive does not return control to RT-11; rather, it stays in an infinite loop. The overall algorithm is given below:

```
Preset all Common Data
Open all files
Initialize HP Terminal
Schedule Menu Manager to display initial start menu
Do forever
```

```
        Get next BG Task identifier from Control I/O Manager
        Call BG Task
    Close all files
```

Thus the Executive receives the request for the next BG Task from the Control I/O Manager and then transfers control to the selected BG Task. The following paragraphs discuss the Common Routines.

1. <u>Common Routine ´endfbk´</u>

Feedback to any operator´s input is done in the following way. There is always an acoustic feedback. Permanent buttons on the screen are displayed in a reverse mode if the related function is active. Other buttons are displayed in reverse mode after they have been selected by the operator. The permanent buttons for hardcopy are displayed in a reverse mode for about one second. If these functions are activated, the Input Manager reverses the button display and schedules this Common Routine to be executed one second later. It then will display the button in the normal mode again.

2. <u>Common Routine ´enhanc´</u>

This routine performs an enhancement of a displayed button on the screen. It retrieves the necessary button coordinates from the input options and sends a command string to the HP terminal to perform the requested enhancment.

3. Common Routine ´errmsg´

This routine was implemented for two reasons. (1) The Message Manager which is responsible for display of any operator input related messages or warning messages is a completion routine and may not be called by programs executing as a BG mainline program (e.g. BG Task). Thus the messages have to be stored temporarily until the Message Manager is called. (2) There is only one routine processing all requests to display a message.

There is a buffer for only one input-related message; the Message Manager will be invoked to display the message before the next operator input is possible. Warning messages may be created in a larger number until the next call of the Message Manager. Thus the variable length messages are stored in a buffer large enough to save 280 characters of text.

4. Common Routine ´ascint´

All the information on the Menu File is coded in ASCII. In order to store a three digit integer number in two bytes (one word), such a number is converted using the following transformation:

Upper Byte = integer/30 + 40

Lower Byte = integer modulo 30 + 40.

This ensures that both bytes will contain valid ASCII characters. This routine will convert such a coded number into an integer.

D. CONTROL I/O MANAGER

This is a BG mainline program. It is called by either the Executive or any BG Task to process a request for input from the operator. These calls are mutually exclusive; either the Executive is executing or it has transferred program control to a BG Task. The Control I/O Manager returns only when a valid operator input has been made. Thus it remains looping until the Input Manager provides a valid input. If a BG Task requests that a new menu has to be displayed, the Control I/O Manager will schedule the immediate call of the Menu Manager.

E. INPUT MANAGER

This program executes on completion routine level. Thus it will not be interrupted by any BG Monitor program. It is initially called by the Executive and then it re-schedules its call every 0.1 seconds by using an RT-11 programmed request. It examines the comman data ´flag´ to check whether any operator input has ocurred.

Upon an input from the operator the Log Manager will be called to log the command if requested and feedback will be provided. If the chosen input option requests a new menu to be displayed the Menu Manager will be scheduled to display it. Otherwise the currently displayed menu will be erased one second later. In case of an input from a permanent button the Control Command Manager is called. Note that all programs called by the Input Manager also execute on completion routine level.

F. CONTROL COMMAND MANAGER

This program processes all inputs from permanent buttons or any controller menu buttons. The permanent buttons are:

- PRINT CRT1:	Make a hardcopy of the graphics display on CRT 1
- PRINT CRT2:	Make a hardcopy of the graphics display on CRT 2
- STATUS:	Display the system status
- SELECT:	Select a new function
- CONTROL:	Change SSA control parameters
- HELP:	Provide help to the operator with respect to further actions
- ACKNOWLEDGE:	Acknowledgement of a displayed warning mesage; erase it

The permanent buttons Status, Select, Control, and Help function in an on/off mode. When they are chosen the first time, the function is initiated, the menu will be displayed, and the permanent button will become enhanced. When an active function (permanent button is enhanced) is chosen, the function will be deactivated, the button will be displayed normally, and any waiting menu or the initial start menu will be displayed.

If a new BG Task is to be started, any old Task menu will be erased and the identifier for the new BG Task is saved to be returned to the Executive.

Any input request to change the current PB will be processed by the Parameter Block Manager.

C. MENU MANAGER

This program is the most important part of the BG Controller. It executes on completion routine level and is responsible for updating the display on the HP Terminal and updating the valid input options.

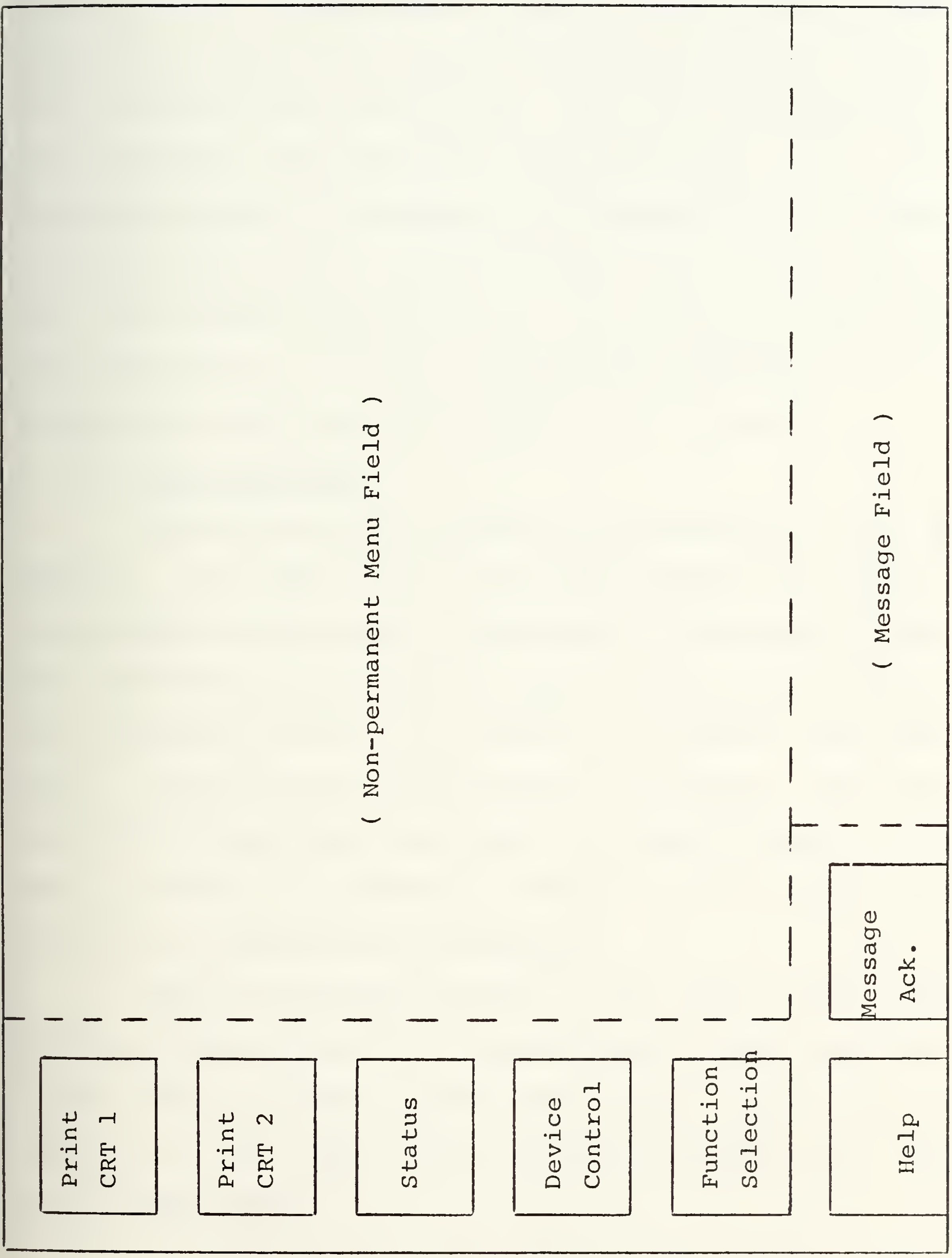
1. Menu Display

This paragraph gives a description of the display layout used to display menu information. There are three different areas on the screen. (1) The permanent buttons are displayed as a column at the left side of the screen. They will never be changed during program execution. (2) The bottom lines of the screen are reserved to display operator input related messages and warning messages. (3) The remaining field is used to display any menu initiated by the BG Controller or any BG Task. Fig. 5.2 displays the layout.

When a new menu is being displayed, the menu field will be erased first. To erase the image on a line, the image can be overwritten with blank characters (quite slow) or it can be erased up to the end of the line. The latter method is faster and has been implemented. This is the reason why the permanent buttons are displayed on the left side of the screen.

As the permanent buttons always are displayed at the same locations, the operator, once familiar with the layout, can easily locate these buttons.

Figure 5.2
HP Terminal Display Layout



2. <u>Menu Stack ´menstk´</u>

The BG Controller allows the operator to have initiated several functions at a time. Thus the Menu Manager has to keep track of all functions and their related menus. This is done using a menu stack. The top of this stack always holds the controller menu being displayed. If a new controller menu is requested, it is placed on the stack and it will be displayed. The menu stack holds 20 menu identifier. If a stack overflow occurs, it is saved on disc in the menu stack file. In case of a menu stack underflow, an existing stack from the menu stack file is retrieved.

3. <u>Menu Hierarchy</u>

There exist three different kind of menus in the BG Monitor. (1) The Initial Start Menu contains the image of the permanent buttons and is displayed at the program start. (2) The display of Controller Menus is initiated by selecting a permanent button or a button of another Controller Menu. (3) Task Menus are initiated by BG Tasks. There exists only one Task Menu at a time in the Menu Manager. Its menu identifier is stored in ´icurtm´. It is either being displayed or waiting to be displayed.

The Menu Manager displays a new menu using this priority scheme: First any Controller Menu in the menu stack is selected. If there is none, any waiting Task Menu is considered. If there is none, the Initial Start Menu will be displayed again.

4. <u>Menu Structure</u>

All menus are organized in a tree structure. Each menu provides information about its predecessor menu and each of its buttons may request a successor menu. Thus it is possible to always go up and down within this menu tree. The Menu Manager keeps track of all paths within this tree structure that have been selected by initiation of one or more functions. Actually only the menu at the bottom of a particular path has to be saved. A predecessor menu always can be determined uniquely. Its menu-id is part of the information provided by a menu and is stored for all menus being displayed in ´ipredm´.

The menu stack ´menstk´ saves the menu-ids of all menus at the bottom of a path. Thus a new menu of the same function just replaces the predecessor menu on top of the menu stack. A new menu of a different function is placed on top of the stack. If an active function is deactivated (erasing of the menu at the bottom of a path), the next menu in the stack will invoke that function having been active previously.

This method provides the possibility for an operator to initiate as many functions as he wishes. This allows him to have several functions at intermediate steps at the same time. In addition, a BG Task may run and perform a variety of actions that do not require operator inputs.

Many activities within a function only require the display of another menu. Thus a great deal of logic and de-

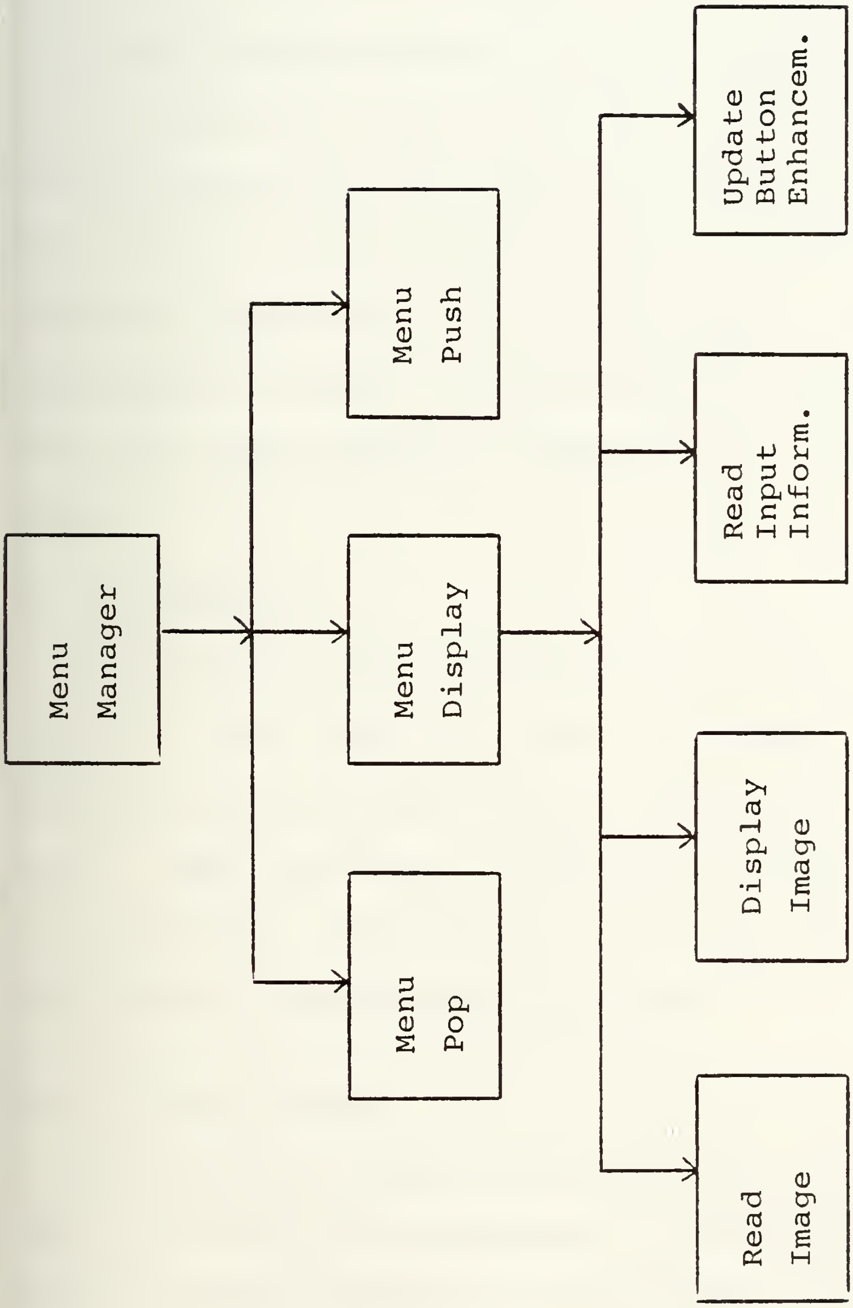
cisions is built into the menu structure. This permits a simpler program logic and shifts many changes of the BG Monitor towards an update of the menu file, which requires no program modification.

5. Functions

Depending on the requested function and the status of the menu stack these related actions will follow: (1) Update of the menu stack, (2) retrieval and display of the menu's image, (3) retrieval and storage of the input options, and (4) update of any button enhancements. Fig. 5.3 displays the overall program structure.

Figure 5.3

Structure of the Menu Manager



H. PARAMETER BLOCK MANAGER

The actions of the FG Monitor are based on information stored in the Parameter Block (PB). These parameters may be modified by the operator independent of any BG Task running. The related operator interaction is processed by the Parameter Block Manager.

The PB Manager saves all new parameters in the PB. After completion of the update, the PB is sent to the FG Monitor using the communication services provided by RT-11. The PB must not be sent to the FG Monitor before the whole update is completed. This could result in invalid interrelationship between the parameters available to the FG Monitor (i.e. some parameter changes have to be done simultaneously).

I. LOG MANAGER

When requested all operator´s inputs will be logged and saved in the log file. The plain-English log text can be retrieved from the input option information stored in common data. The Log Manager sequentially writes these log texts onto the disc. The contents of the log file can be printed to trace all the operator´s actions.

K. MESSAGE MANAGER

The Message Manager is responsible for updating the message field and the enhancement of the ACKNOWLEDGE button on the HP Terminal display. It processes requests by either the BG Controller or any BG Task to display or erase mes-

sages. These messages may be initialized by an operator´s input or by any failure detected by the BG Monitor. These two types are handled differently. Only one operator input related message is expected to be existent when the Message Manager is called by RT-11. It is assumed that the operator has no chance to do another invalid input before the Message Manager processes the first illegal input.

Also it is assumed that a BG Task only generates one warning message before the Message Manager will be called. This is because a BG Task executes as a mainline program. If it detects an error it schedules the Message Manager, a completion routine, at once. Thus the Message Manager will be called and the BG Task´s execution will be interrupted. Warning messages generated by the BG Controller, however, may be generated when executing on completion routine level. As these completion routines do not interrupt each other, there may be more than one warning message to be processed by the Message Manager when it is called.

1. <u>Input Related Messages</u>

When any program detects an illegal operator input it places a message text in ´inpmsg´ and schedules the Message Manager to be executed at once. The Message Manager then displays this message on the related line within the message field on the screen. Any other input related message being displayed will be overwritten. Thus the message always responds to the last operator´s input.

Any input related message will be erased when the

operator does any input. The Input Manager always schedules the Message Manager to erase any existing input related message when a valid input has been made. In case of an illegal input the old message will be overwriten by a new one.

2. Warning Messages

The text of a warning message generated by a BG Task will be in ´iwmsg´. Warning messages generated by the BG Controller are saved in ´msgw´.

If no warning message is being displayed the new message will be written on the screen. If there is already a warning message being displayed the new one will be saved on the message stack. In case of a stack overflow it will be saved on disc in the message stack file. The Message Manager will retrieve all stacked messages using the first-in first-out (FIFO) scheme. The message stack is divided into a lower and an upper half. A read index points to the next message to retrieve from the stack; a write index points to the next free entry in the stack. The read index always stays in the lower half of the stack. If only the lower stack half is used, the write index stays in this half. Else it always points to entries in the upper half.

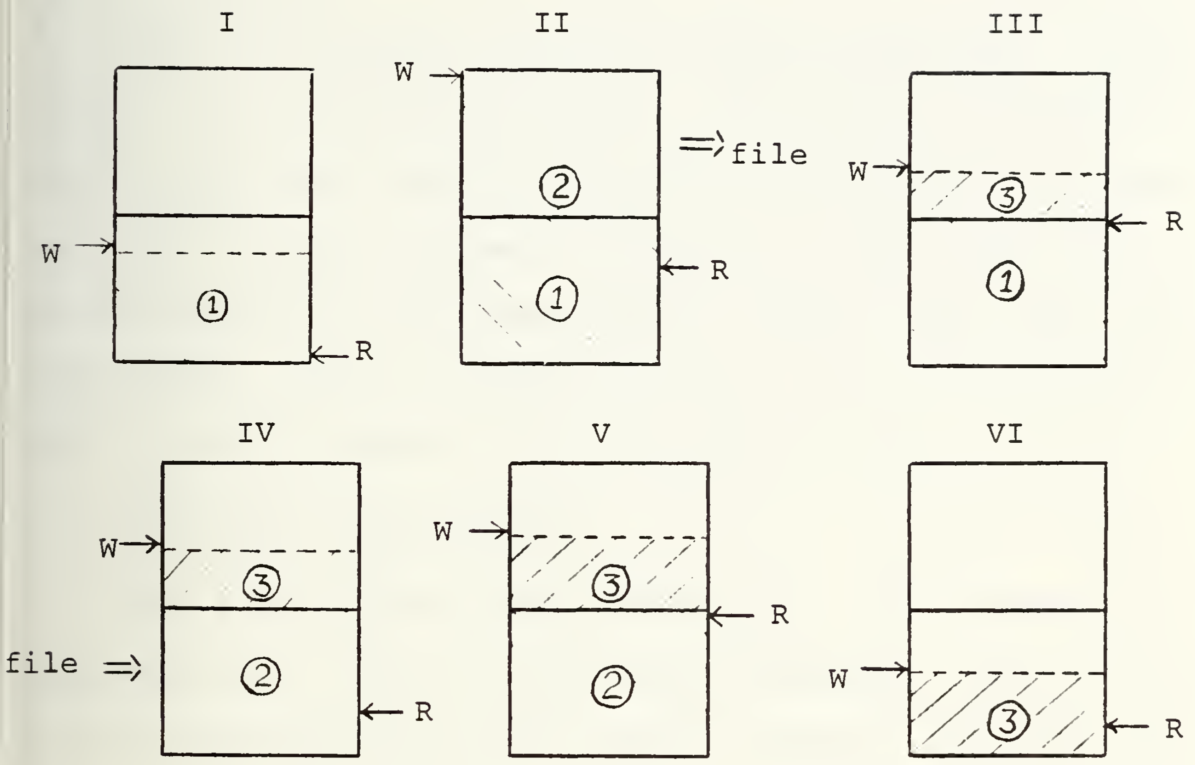
When the upper half is filled, it is saved in the Message Stack File and the write index is reset to the beginning of the upper half. When the read index reaches the end of the lower half, the lower half is filled either with a saved stack in the file or with the upper half. The read index is reset and in the latter case the write index points

to an entry in the lower half. Fig. 5.4 illustrates an example.

Any saved warning message will cause a blinking of the ACKNOWLEDGE button. Upon selection of the ACKNOWLEDGE button the operator may initiate erasing of a displayed warning message. The Message Manager will display any other waiting message from the message stack or clear the message line on the screen. If there are no more warning messages waiting on the message stack the ACKNOWLEDGE button will stop blinking.

Figure 5.4

Message Stack Illustration



Phases:
- I - Only lower stack half in use
- II - Upper half has been filled and is written on disc; write will continue
- III - lower half is read; read will continue
- IV - stack half is retrieved from disc; read will continue
- V - lower half has been read;read will continue
- VI - Upper half has been moved into lower half

Legend: ⓧ - Sequence of message blocks

R - Read index; W - Write index

VI. BACKGROUND TASKS

A. INTRODUCTION

A BG Task performs a group of related functions. It executes as a BG mainline program. The BG Tasks are independent of each other and only one will be executed at a time. Thus the overlay structure of RT-11 is used. When linking the BG Monitor all BG Tasks will be assigned to the same memory area. When a certain BG Task is called by the Executive it will be loaded if not already in memory.

B. BACKGROUND TASK STRUCTURE

After a BG Task has been started it will execute independently of other BG Controller functions. Only if the BG Task needs some operator input to proceed it will call the Control I/O Manager. Also it may schedule the call of the Message Manager to display or erase a message.

The operator has the option to start a new BG Task while another is executing. Thus the old one has to be killed. The Executive will set a common data unit, the ´KILL-flag´, which indicates that the running BG Task is to be killed.

To recognize this condition the BG Task has to check this flag. Thus the design of a BG Task has to be done in such a way that the control path always passes certain check points to examine the KILL flag. When the KILL condition is recognized the BG Task must not initialize any new I/O or schedule any other program. It returns control to the Executive after all initialized I/O has been completed. Ap-

propriately the KILL condition is checked after a specific sub-function has been completed or while waiting for I/O completion.

VII. SUPPORT SOFTWARE

A. INTRODUCTION

Most of the BG Monitor's actions are determined by the contents of the menu file. This file contains all information to create an image on the HP Terminal and to process the operator's inputs from keyboard or Touch Input System. It is generated and updated using Support Software that executes under the control of UNIX.

One program updates the Menu File, it is called Menu Editor. Another program, the Menu Checker, is used to verify any modifications of the menu file.

B. MENU EDITOR

1. Introduction

The Menu Editor processes fixed format input and updates the menu file. To simplify the menu generation the user does not have to specify the exact locations of text or displayed buttons on the screen; the Menu Editor will perform all necessary arrangements of the image.

The Menu Editor needs three files which have to exist in the UNIX file system:

- menu file,
- intermediate menu file,
- input file.

The input for the Menu Editor can not be taken from a standard input device because the program reads over the input file twice. The first time all requested information

will be extracted to compute the image size, to get the referenced menu names, to update the menu file directory, and to provide the Menu Editor with sufficient knowledge to arrange the created image. During the second read over the input file a new or updated menu will be the result.

The following terminology will be used in the further presentation:

- image:	the complete image of the HP Terminal
- image block:	a subset of the image; it is one block of text or it is a question with its responses
- response:	subset of a question block; it is a key or a displayed button
- predecessor menu:	a menu which preceeds some other menu in the display sequence
- successor menu:	a menu which succeeds some other menu in the display sequence

The Menu Editor Manual in the appendix will give sufficient information about how to use the program.

2. Files

The Menu File contains a Menu File Directory and the menus. Its records have a fixed length of 544 bytes. The directory uses 20 records with 45 entries each. For a menu,

6 records are needed to store the maximum possible amount of information. Because UNIX does not provide direct access I/O in FORTRAN, the Menu File has to be read sequentially. The UNIX I/O system does not write leading or trailing ASCII NUL characters. Therefore it is crucial that the bytes 1 and 542 of a record do not contain such a character. Another constraint by UNIX prohibits more then fifteen open requests for files. Thus a record that is located between the beginning of a file and the read pointer can only be accessed by closing and reopening the file. This is the only way to reset the file´s pointers. This procedure easily exceeds the maximum number of file open requests. Therefore a Temporary Menu File, ´menuout´, is used.

There are two reads over the Menu File. The first time the locations of referenced menus in the menu file are extracted. Also the directory is updated and the new version is copied to the Temporary Menu File. The second time all unchanged menu records are copied and updated records are written to the Temporary Menu File. Finally the whole Temporary Menu File is copied to the Menu File.

When creating the menu records each question or text section is considered to be an image block. Such a block will always have the same structure depending on the input. Text simply will be copied in the demanded form. The image commands for a question will be computed depending on the location of the buttons, their size, and their number. When the number of required lines on the screen is determined the

whole image will be rearranged by separating the blocks with available blank lines.

3. <u>Functions</u>

The Menu Editor will perform several functions which all update the menu file directory and modify menu records.

a. Initial Start Menu

The first menu that has to be created is the Initial Start Menu. This function initializes the Menu File and creates entries in the directory for all initial menus, like Initial Status Menu, Initial Select Menu, and so on. The BG Monitor expects these menus at fixed record locations.

The records of the Initial Start Menu are created. The image records will contain the command for the HP Terminal to (1) draw all permanent buttons on the screen, and (2) to clear the rest of the screen. Also the information about possible input options is stored.

b. Create a Menu

This function is used to create a new menu. The first empty entry in the menu directory will be filled with the new menu name and the number of its first record. If the menu´s predecessor menu is not yet existent a new entry will be generated for it. If the menu has any successor menus their entries will be searched in the directory; if they are not yet existent new entries will be generated. After the directory has been updated, the HP Terminal commands and information about possible inputs will be computed

and stored in the menu records.

c. Update a Menu

This function is only allowed if the referenced menu is already in the Menu File. Its location is retrieved from the menu directory and the menu records are generated the same way as with menu creation.

d. Delete a Menu

This function simply searches the referenced menu´s entry in the menu directory and deletes it. A menu to be deleted may have successor menus. The user has to delete any successor menus if they are no longer needed. The predecessor menu has to be updated as well.

C. MENU CHECKER

1. Introduction

This program allows verification of any modifications of the Menu File. It extracts the contents of the menu directory and the menu records. The function requests are taken from the standard input device. Output is done at the standard output device.

The Menu Checker Manual in the appendix will give sufficient information about how to use the program. Also examples of possible outputs are shown following the appendices.

2. Functions

The following functions may be selected by the user:

- Directory: All entries of the Menu File directory will be extracted

- Input options:	All information about possible input options of a specified menu will be extracted
- Image:	The HP Terminal commands that create the image will be extracted

VIII. BACKGROUND MONITOR FILES

A. INTRODUCTION

The BG Monitor uses several files for different purposes. A Menu File contains all information to control the operator interaction and resulting actions. Two temporary files, the Menu Stack File and the Mesage Stack File are used to save any overflown stacks. A Log File will contain plain-English input commands done by the operator.

B. MENU FILE

This file is created using the Menu Editor. Its mode is direct access and its form is unformatted. All information is represented in ASCII characters. Integers are converted into a 2 character ASCII code using the following mapping:

Upper byte = integer/30 + 40

Lower byte = integer modulo 30 + 40

The file has fixed length records of 544 bytes each. A menu occupies 6 records to store its information. To store the directory 20 records with 45 entries each are used. Each entry contains a menu name and the location of its first data record. The menu name is represented in 10 ASCII characters. The pointer to a menu´s first data record is a 3 digit integer. There exists a one-to-one mapping between the location of a directory entry and the location of the menu records. The following parameters are used:

- i: location of the first menu record behind the directory

- j: location of the directory record
- k: pointer to entry in the directory

The menu´s first data record number is determined by

record number = i + (45*(j-1) + k-1)*6.

The coded record number is only used by the Menu Editor. The BG Controller does not have to access the directory at all. References to a menu by another one is provided by its record number to decrease the number of necessary file accesses.

The first 4 records of a menu are used to store the HP terminal commands which will create an image on the display. Subsets of a command string are separated by an ASCII ´$´. The end of data in an image record is indicated by an ASCII ´|´. The last used image record has an ASCII ´~´ in byte 540. These three ASCII characters therefor must not be used in any image text.

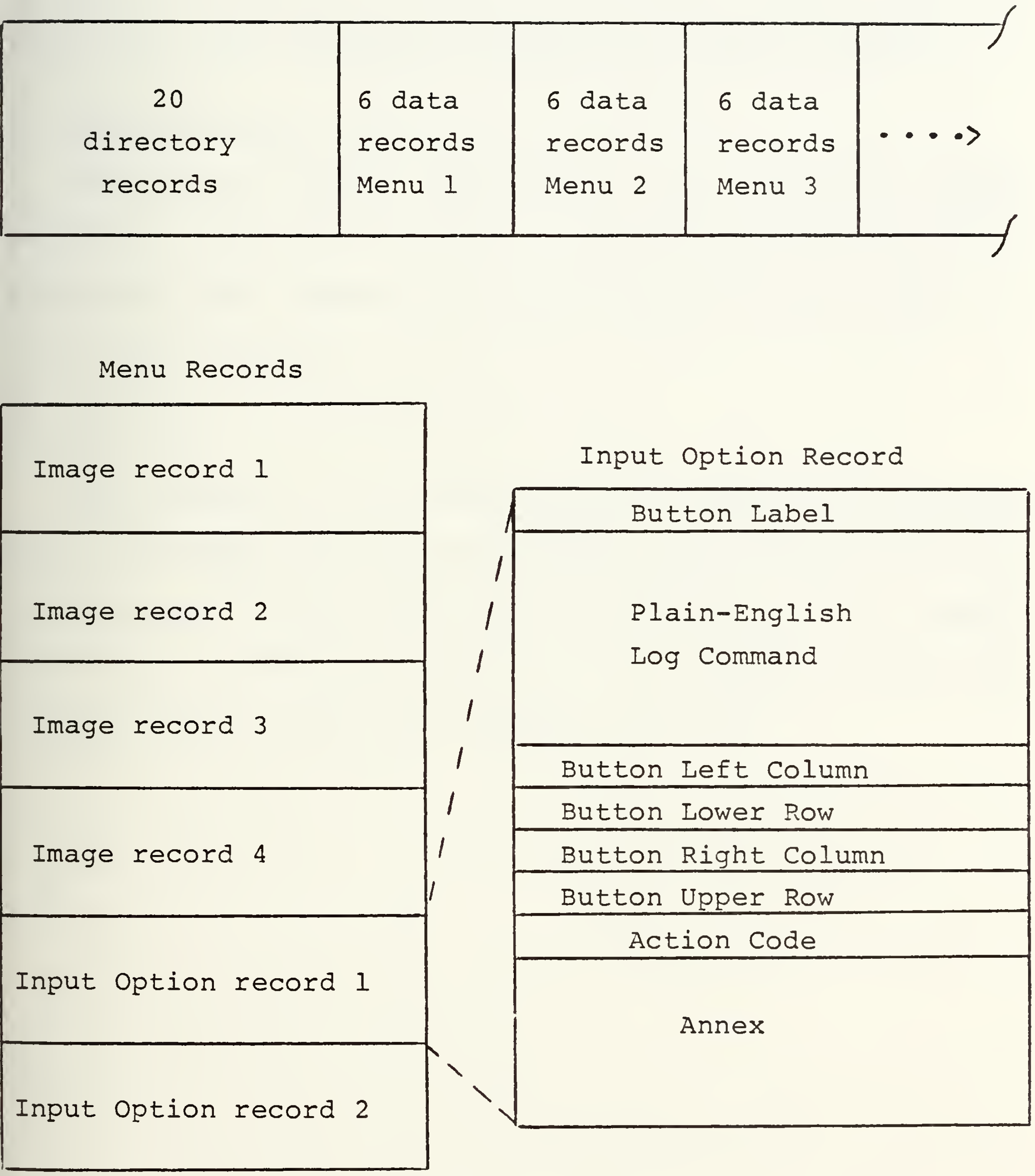
The last two records contain information about possible operator inputs as well as a parameter block. A record is partitioned into 30 byte sections, thus providing entries for 18 input options. These are the data stored for each possible input:

- word 1: Legend of the button or the key
  (ASCII character in lower byte)
- word 2
  to 6: plain-English log command (10 characters)
- word 7: left boundary of a button
- word 8: lower boundary of a button

- word 9: right boundary of a button
- word 10: upper boundary of a button
- word 11: action code; it determines the reaction of the BG Monitor
- word 12 to 15: used for any additional information (e.g. BG Task identifier)

A parameter block informs the Menu Manager about some additional menu parameters. The predecessor´s menu-id is stored. The number of possible input options is given. Also the type of the menu (e.g. Status menu) can be retrieved from this parameter block. Word 15 of this block contains the ASCII characters ´~~´. The parameter block is located in the 36th entry for the Initial Start Menu. For all other menus it is located in the 29th entry. Fig. 8.1 will display the file layout.

Figure 8.1
Menu File Layout



C. MENU STACK FILE

This file is used to intermediately save any menu stacks. It is a direct access, unformatted file. One stack consists of 40 bytes to store 20 menu-ids. Stacks are written to the file and read in a Last-out First-in scheme.

D. MESSAGE STACK FILE

This direct access file saves any overflown message stacks in unformatted form. A message stack has a size of 120 bytes. The scheme to save and retrieve these stacks is First-out First-in.

E. LOG FILE

When logging of all operator input commands is requested the Log Manager writes plain-English commands to this file. It is accessed sequentially and is formatted. Each command exists of 10 ASCII characters and the contents of this file can be listed to trace all operator actions.

IX. CONCLUSIONS

The software for the Background (BG) Monitor documented in this thesis has been developed and tested successfully.

The data in the Menu File causes specific actions to be taken by the BG Monitor when an operator's input is processed. Thus the execution of the BG Monitor is to a large extent controlled by the contents of the Menu File. This control, imbedded in the data, decreased the size of the BG Controller and allowed a general program structure which is very flexible. Many changes of a BG Monitor's function can be implemented by just updating the Menu File. Thus no program modifications are necessary.

Extensions of a certain function or addition of new functions without changing the structure of the BG Monitor are possible. This can be done simply by adding new menus to the Menu File and by installing a new BG Task. This only requires minor changes in the Executive (to call the BG Task) and creation of new menus in the Menu File. A BG Task can be developed and tested separately from the existent BG Monitor.

Further software modifications are possible. The menu record size (now 544 bytes) could be reduced to 512 bytes (standard physical record size on a disc). This would increase the speed of data transfer by approximately 100 milliseconds. The new size would be sufficient to store all information about a specific menu.

Certain I/O problems with the operating systems caused a considerable delay in the software development. UNIX only allows fifteen requests to open files from FORTRAN programs. This restriction is not documented in the UNIX manuals. To cope with this problem, the form of some file operations had to be changed.

A software error related to I/O within completion routines was detected in RT-11. A patch received from the Digital Equipment Corporation was implemented in RT-11 and resolved this problem. Source code for this patch is included with the program listing section of this thesis.

APPENDIX A

SUMMARY OF SOFTWARE AND DOCUMENTATION

This summary lists the existing programs and manuals which have been written during the software development.

- BG Monitor, consisting of the BG Controller and BG Tasks
- Menu Editor
- Menu Checker
- Manual for the Menu Editor
- Manual for the Menu Checker.

APPENDIX B

MENU EDITOR MANUAL

A. INTRODUCTION

The Menu Editor is a program that executes under the operating system UNIX. It is used to create, update, or delete menus in the Menu File used by the BG Monitor of the SATCOM Signal Analyzer. The RATFOR source program is stored in the file ´menued.r´. The executable program is saved in the file ´menued´. The Menu Editor will execute using the command:

```
                menued
```

B. FILES

The Menu Editor requires three different files that have to exist in the UNIX file system. These are their filenames:

- ´menufile´ : This is the Menu File that will be used by the BG Monitor.
- ´menuout´ : This is a temporary menu file used to store intermediate data.
- ´infile´ : This file contains all user specified input to modify the Menu File.

All files will be accesses sequentially, the contents is coded in ASCII.

1. File ´menufile´

If the file initially is not yet existent, it has to be created using the UNIX Editor ´ned´. Any dummy input may

be stored in the file. The structure of 'menufile' will be determined by the Menu Editor when it is accessed the first time.

The file consists of 544 bytes records. Each record stores valid information within the first 540 bytes. The last four bytes will contain the ASCII characters 0, 0, CR, LF. The Menu Editor will write records with a length of 542 bytes. UNIX will add CR/LF, thus creating a record of 544 bytes.

The first 20 records will contain the Menu File directory. Each directory record allows 45 entries. An entry contains a menu name (10 ASCII characters) and a pointer to the first data record of that menu. The initial directory will be set up when executing the function 'Create Initial Start Menu'.

For each menu 6 records will be used. The first four records contain the commands to create an image on the HP Terminal display. Subsets of command strings are separated by ASCII '$'. The end of usable image commands on a record is indicated by an ASCII '|'. The last used image record will contain an ASCII '~' in byte 540. These three ASCII characters must not be used in any menu's image text.

Two additional records provide information about possible operator inputs in reaction to the display of this menu. These records are partitioned into 30 bytes sections, thus providing entries for 18 input options each. The information stored for each input option will be

- Label of a button or key
- plain-English log command
- boundaries of a displayed button (lines and columns)
- an action code to determine the BG Monitor´s action
- additional information that may be requested

A parameter block within these records will contain additional information about some menu parameters. It stores (1) the predecessor´s menu-id, (2) the menu´s type (e.g. Status Menu), and (3) the number of possible input options. Word 15 of this entry will contain the ASCII characters ´~~´. The parameter block is located in the 36th entry for the Initial Start Menu. For all other menus it is located in the 29th entry.

2. File ´menuout´

If this file does not yet exist in the UNIX file system, it has to be created using the UNIX Editor ´ned´. It may contain any data. This file will have exactly the same structure as the Menu File. It is used to copy unchanged records from ´menufile´ and to add updated records. When the Menu Editor has completed its requested function, it copies the contents of ´menuout´ to the Menu File

3. File ´infile´

This file will provide all information necessary to run the Menu Editor. The file is to be created using the UNIX Editor ´ned´. The following terminology will be used

in this manual:

- predecessor menu:	a menu which preceeds some other menu in the display sequence;
- successor menu:	a menu which succeeds some other menu in the display sequence;
- image:	the image on the HP Terminal display created by the menu
- image block:	a subset of the image; it is one block of text or it is a question with its responses;
- response:	subset of a question block; it is a key or a displayed button;

All integer inputs have to be ended with ´,´. Character inputs have to be ended with ´@´. A line of text is to be ended either by ´|´ (if it is not the last text line) or by ´@´. Comments are allowed on each line following these ending symbols.

a. Format of ´infile´

The input format is given below. A number in parentheses indicates the line number in ´infile´.

```
    for all functions...
    (1) : function code
    for delete, create, update...
    (2) : menu name
    for create, update...
    (3) : predecessor menu name
    (4) : menu type
    (5) : menu title
    (6) : number of image blocks
        for each image block...
        : start-block code (126)
        : number of responses
        : location of buttons
        : size of buttons
        : text lines
            for each response...
            : start response code (127)
            : label of button
            : log command
            : action code
            : annex
```

b. Contents of ´infile´

This section will specify which values are valid for all possible inputs.

function code:

1 - DELETE an existing menu

2 - CREATE a new menu

3 - UPDATE an existing menu

8 - CREATE INITIAL START menu

menu names, log command:

10 characters

menu type:

1 - INITIAL START menu

2 - STATUS menus

3 - SELECT menus

4 - HELP menus

5 - CONTROL menus

6 - BG TASK menus

menu title:

text for first line on top of menu image

location of buttons:

1 - button row below the text

2 - button row beside the text

3 - button column beside the text

size of buttons:

number of lines the button will cover

text:

one or more lines of text

label:

1 character

action code:

1 - display a new menu

3 - return an integer

4 - return a character

5 - display the previous menu

7 - HARDCOPY CRT1

8 - HARDCOPY CRT2

9 - ACKNOWLEDGE

10- STATUS

11- SELECT

12- HELP

13- CONTROL

14- New BG Task

15- Change Parameter Block

annex:

action code 1: menu name

action code 3: integer

action code 4: character

action code 14: BG Task identifier

c. Example

This section will give an example of how the ´infile´ might look like. This input causes the Menu Editor to create a new menu which has one question block with two responses and a pure text block.

```
2,                          create a new menu
testmenu  @                 name of the new menu
startmenu @                 predecessor menu
6,                          menu type: Task menu
first Task menu@            text of title on top line
2,                          2 image blocks
126,                        start-code for first block
```

```
2,                          2 responses

3,                          button column beside text

3,                          button size 3 lines

This is the text of the question|        1st line

    If you want to continue touch ´Y´|   2nd line

    else touch button ´N´@               3rd line

127,                        start-code 1st response

N@                          label of button

No          @               log command

3,                          action: return integer

20,                         integer to return

127,                        start-code 2nd response

Y@                          Label of button

Yes         @               Log command

1,                          action: display new menu

nextmenu    @               menu name

126,                        start-code 2nd block

0,                          no responses

0,                          no buttons

0,                          no button size

This is a pure text block|

It just gives some information and has|

no input options@
```

C. SEQUENCE OF MENU GENERATION

The initial BG Controller menus have to be located in fixed Menu File records. This is requested because the BG Controller of the SSA has to find these menus. These are the associated menu records:

1) INITIAL START menu	-- 21
2) initial STATUS menu	-- 27
3) initial SELECT menu	-- 33
4) initial HELP menu	-- 39
5) initial CONTROL menu	-- 45

To guarantee this sequence the INITIAL START menu has to be generated first. The Menu Editor then will set up the Menu File directory, create the initial start menu, and place the other initial BG Controller menus into the directory. The implementation of these menus then has to be done using the UPDATE function.

D. FUNCTIONS

1. CREATE INITIAL START menu

This is the very first function the Menu Editor has to process. It will build the Menu File directory and place the entries of all initial menus into the directory. The menu records for the INITIAL START menu will be created and stored following the directory.

2. CREATE a menu

The Menu Editor searches the menu direcory and uses the first empty entry to store the menu´s name and pointer to its records. If the menu´s predecessor menu is not yet

existent its name and computed pointer will be placed into the directory. If the menu has any successor menus, they will be searched in the directory, if they are not yet created, new entries for them will be generated.

After updating the directory and extracting needed data from it the image commands and information about possible input options will be generated and stored in the menu´s records.

3. UPDATE a menu

To process this function, the referenced menu has to be in the ´menufile´. Essentially the further processing is the same as with the CREATE function.

4. DELETE a menu

The menu´s entry is searched in the menu directory. Then the menu´s name will be deleted. A deleted menu may be a predecessor or successor of other menus. The user has to update any other menus that are affected by the deletion.

E. MENU CHECKER

There exists a program ´MENU CHECKER´ that allows to verify any modification of the ´menufile´. It has the following functions:

- DIRECTORY:	Extract all entries from the Menu File directory
- INPUT OPTIONS:	Extract all information about possible input options for a specific menu
- IMAGE:	Extract the image generating

commands for a specific menu

For further information refer to the MENU CHECKER MANUAL.

APPENDIX C

MENU CHECKER MANUAL

A. INTRODUCTION

The Menu Checker allows to verify the contents of the Menu File. This program executes under the operating system UNIX and exists as an executable file. The file ´mcheck.r´ stores its source code. The Menu Checker will execute using the command:

```
mcheck
```

It offers the following functions to extract data from ´menufile´:

- DIRECTORY:	Extract all entries from the Menu File directory
- INPUT OPTIONS:	Extract all information about possible input options for a specific menu
- IMAGE:	Extract the image generating commands for a specific menu

The Menu Checker needs to have access to the ´menufile´. Input is taken from the standard input device, output is directed to the standard output device. Non-printing ASCII characters of the image commands (e.g. Escape) will not appear in the output.

B. INPUT FORMAT

The input to control the Menu Checker consists of one or two parameters. An integer has to be ended with an ´,´ and

a 10-character menu name has to be ended with ´@´. The following example shows the input format and valid parameters. The input sequence is indicated by the numbers in parentheses:

```
(1) : function code
                              5 - directory
                              6 - image
                              7 - input options
(2) : menu name for functions 6,7

example:
      6,             extract image commands
      startmenu @    from startmenu
```

SAMPLE OUTPUTS OF THE MENU CHECKER

```
These are the directory records


record number:  1
startmenu   21   status0     27   select0     33
help0       39   control0    45   status1     51
task1       57   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0
0000000000   0   0000000000   0   0000000000   0

record number:  2
There are no entries

record number:  3
There are no entries

record number:  4
There are no entries

record number:  5
There are no entries

record number:  6
There are no entries

record number:  7
There are no entries

record number:  8
There are no entries

record number:  9
There are no entries

record number: 10
There are no entries

record number: 11
```

```
There are no entries

record number: 12
There are no entries

record number: 13
There are no entries

record number: 14
There are no entries

record number: 15
There are no entries

record number: 16
There are no entries

record number: 17
There are no entries

record number: 18
There are no entries

record number: 19
There are no entries

record number: 20
There are no entries
```

Image commands:

```
a07C&a01Ca01r01CdJd@
a02r01CdJd@
a03r01CdJd@
a02r02CPR 1
a07C&a01Ca05r01CdJd@
a06r01CdJd@
a07r01CdJd@
a06r02CPR 2
a07C&a01Ca09r01CdJd@
a10r01CdJd@
a11r01CdJd@
a10r02CSTAT
a07C&a01Ca13r01CdJd@
a14r01CdJd@
a15r01CdJd@
a14r02CCTRL
a07C&a01Ca17r01CdJd@
a18r01CdJd@
a19r01CdJd@
a18r02CSEL
a07C&a01Ca21r01CdJd@
a22r01CdJd@
a23r01CdJd@a22r02CHELP
a16C&a10Ca21r10CdJd@
a22r10CdJd@
a23r10CdJd@a22r11CACK
a01r12C
```

```
Input options for menu:   startmenu
first record: 21

number of input options:  7
menu type:  1
predecessor menu: 0000000000


input option  1
label: P
log command: PRINT CRT1
button left collumn:  1
button right collumn:  6
button upper row:  1
button lower row:  3
action code:  7


input option  2
label: D
log command: PRINT CRT2
button left collumn:  1
button right collumn:  6
button upper row:  5
button lower row:  7
action code:  8


input option  3
label: A
log command: ACKNOWLEDG
button left collumn: 10
button right collumn: 15
button upper row: 21
button lower row: 23
action code:  9


input option  4
label: S
log command: STATUS
button left collumn:  1
button right collumn:  6
button upper row:  9
button lower row: 11
action code: 10


input option  5
label: E
log command: SELECT
button left collumn:  1
button right collumn:  6
button upper row: 17
```

```
button lower row: 19
action code: 11


input option  6
label: H
log command: HELP
button left collumn:  1
button right collumn:  6
button upper row: 21
button lower row: 23
action code: 12


input option  7
label: C
log command: CONTROL
button left collumn:  1
button right collumn:  6
button upper row: 13
button lower row: 15
action code: 13
```

BACKGROUND CONTROLLER PROGRAM

```
#
#
#
#
#**********************************************************************
#**********************************************************************
#                                                                     *
#      B A C K G R O U N D   C O N T R O L L E R
#                                                                     *
#**********************************************************************
#**********************************************************************
#
#      This program is the resident part of the Background Monitor
#        running under RT-11.
#      Additional Background Tasks are stored on a memory image file.
#        When a call is made to such a Task, the RT-11 overlay handler
#        reads it from the memory image file into the specified overlay
#        region.
#
#      The Background (BG) Controller has the following components:
#      - EXECUTIVE
#        This is the Main Program of the BG Controller. It is called by
#        RT-11 to start the execution of the BG Monitor.
#
#      - CONTROL I/O MANAGER
#        This program is called by either the EXECUTIVE or any Task
#        to process an operator input request from the operator
#        keyboard or the touch panel.
#        Control is returned after a requested input has been received
#        from the operator.
#
#      - INPUT MANAGER
#        This program is called initially from the Executive and then
#        re-schedules itself periodically.
#        It processes all operator inputs from either the keyboard
#
#      - CONTROL COMMAND MANAGER
#        This program processes all operator inputs not requested by
#        a BG Task.
#        or the touch panel.
#
#      - MENU MANAGER
#        This program is responsible for updating the display on the
#        operator's console and for providing the other parts of the
#        BG Controller with sufficient information of any possible
#        inputs.
#        The required information will be stored on a separate
#        menu-file.
#
#      - LOG MANAGER
```

```
          This program logs plain-English commands, numbers, and text
          on a requested output device.

        - PARAMETER BLOCK MANAGER
          This program updates the current Parameter Block, saves it
          on disc, and transfers it to the Foreground Monitor.

        - MESSAGE MANAGER
          This program manages the displaying and erasing of any
          operator input related messages or h/w warning messages.

*********************************************************************



*********************************************************************
                                                                    *
              0.  E X E C U T I V E
                                                                    *
*********************************************************************

 Initial Release -- hn/23 Aug 79
 NAME:
       bgexec
 FUNCTION:
       Main program of the BG Controller
       The EXECUTIVE provides common routines for
         the BG Controller
 CALLING PROGRAM:
       Operating System RT-11
 NOTES:
       This program does not return control to the calling program.
 ALGORITHM:
       bgexec:
       open all files used by the Controller
       call all completion routines
       initialize HP Terminal Display
       initialize Carroll Touch Input System
         and save them for future use
       do forever
               { get next task to call from CONTROL I/O MGR;
                 call next task;
               }
#####################################################################


       # parameter for menu table declarations
efine  INMAX   36      # max input options per menu
                       # (including control inputs)
efine  OPTL    15      # storage length per input option
                # ** TBLMAX = INMAX * OPTL
                # ** TBLMAX >= 2*MRL
efine  CONMAX  7       # max number of permanent
```

```
                        # control input
define  TBLMAX  540     # max length of menu table
define  IMAGEX  270     # index in mentbl where the
                        # temporary image storage starts
                   # ** IMAGEX = TBLMAX / 2
define  TEXTL   40      # max word length of text input
define  TNL     10      # max word length of task name

define  MSGL    56      # max word length of displayed message

define  MSGWL   140     # word length of array msgw
define  ENDOFM  126     # code for end-of-message used in
                        # array msgw and msgstk
define  ENDOFI  124     # end-of-input code
                        # used in menu records
define  ENDOFR  126     # end-of-record code to indicate
                        # that a physical record is the last
                        # one of a logical menu record.

        # word length of TT display buffer
define  TTBUFL  80      # 80 characters
define  INTTL   20      # input buffer length from TT

        # Input parameter for subroutine errmsg
define  INM     1       # store input related message
define  WM      2       # store warning message

        # Logical unit number definitions
define  LUMSG   10      # Message-stack file
define  LUMSTK  11      # Menu-stack file
define  LUMENU  12      # Menu file
define  LUTTD   13      # Touch Terminal Display
define  LUTSYS  14      # Touch System
define  LULOG   15      # Command Log File
define  OUT     6       # test output

        # length of physical menu records
define  MRL     270     # word length of physical record
define  MRBL    540     # byte length of physical record

        # Max number of records used by one menu
define  IMGREC  4       # Max number for image
define  OPTREC  2       # Max number for input options

        # Stack size parameter
define  MENSTX  20      # Max menu stack size
define  MGSTL1  120     # Max warning message
                        # stack size
define  MGSTL2  60      # half message stack size
                        # MGSTL2 = MGSTL1 / 2

        # Input parameter definitions for MENU MGR
define  DIMEN   1       # display initial start menu
define  DSMEN   2       # display status menu
```

```
lefine  DSEMN    3        # display select menu
lefine  DHMEN    4        # display help menu
lefine  DCMEN    5        # display control menu
lefine  DTMEN    6        # display task menu
lefine  ECMEN    7        # erase controller menu
lefine  ETMEN    8        # erase task menu

        # Input parameter definitions for MESSAGE MGR
lefine  EINPUT   9        # Erase input related msg
lefine  EWARN    10       # Erase warning msg
lefine  DINPUT   11       # Display input rel. msg
lefine  DWARN    12       # Display warning msg from controller
lefine  DWARNT   13       # Display warning msg from task

        # Action codes from menu file
        # These must not be changed
lefine  ACODEM   1        # display new menu
lefine  ACODET   2        # return text buffer
lefine  ACODEN   3        # return integer
lefine  ACODEC   4        # return character
lefine  ACODEP   5        # display previous menu
lefine  ACODP1   7        # PRINT CRT1
lefine  ACODP2   8        # PRINT CRT2
define  ACODAC   9        # ACKNOWLEDGE
define  ACODST   10       # STATUS
define  ACODSE   11       # SELECT
define  ACODHP   12       # HELP
define  ACODCN   13       # CONTROL
define  ACODTK   14       # New BG Task
define  ACODPB   15       # change PB

        # Input parameter definitions for common
        # subroutine enhanc
define  REV      1        # Reverse button
define  BLKBEG   2        # Start blinking
define  BLKEND   3        # Stop blinking
define  REVEND   4        # stop reverse button

        # Symbolic menu type definitions
        # These must not be changed
define  START    1        # Start menu being displayed
define  STATUS   2        # Status menu being displayed
define  SELECT   3        # Select menu
define  HELP     4        # Help menu
define  CONTROL  5        # Control menu
define  TASK     6        # BG Task menu

        # Symbolic indices for inopt word locations
define  KEY      1        # keyboard entry code
define  PLAIN    2        # Plain-English command
define  LX       7        # Lower left x-coordinate
define  LY       8        # Lower left y-coordinate
define  RX       9        # Upper right x-coordinate
define  UY       10       # Upper right y-coordinate
```

```
efine  CODE     11       # Action code
efine  ANNEX    12       # Additional info

       # Symbolic parameter values
efine  OK       1
efine  NOTOK    0
efine  DONE     1
efine  NODONE   0

       # Symbolic names for flags and parameters
efine  KILL     1        # killfl
efine  NOKILL   0        # killfl
efine  MENU     1        # input control i/o mgr
efine  NOMENU   0        # input control i/o mgr
efine  NOACTN   0        # input completion routines
lefine ACTION   1        # input input mgr
lefine ACCEPT   1        # inflag
lefine NOACPT   0        # inflag

       # Symbolic cursor coordinates
lefine M1X      21       # start x of first message
lefine M1Y      21       # start y of first message
lefine M2X      21       # start x of second message
lefine M2Y      22       # start y of second message



       # ASCCI constant for initializing HP Touch System
lefine UNC      94       # Uncover code '^'
lefine STP      92       # Stop code    ''
lefine CON      128      # Control code
   # Note: This must not be 0, an unused bit is used (=1)
   # If it were 0, it would be truncated as trailing NUL


       # The menu table is used in the following way:
       # - temporary storage for the image records
       #   using the upper half of mentbl
       #     (start at mentbl(IMAGEX) )
       # - input options for permanent displayed 'buttons'
       #   at the top, and input options for non-permanent
       #   'buttons'

       # A program that accesses the input options, declares
       #  an array  inopt(OPTL,INMAX) and equivalences it with
       #  the common array mentbl.
       #       integer*2 inopt(OPTL,INMAX);
       #       equivalence (mentbl(1), inopt(1,1));

       # Menu Table declaration, number of input options
common /blk1/mentbl(TBLMAX), numopt;

       # Operator input storage declarations
       # text,number,character,pointer for text buffer
```

```
ommon /blk2/intext(TEXTL),innumb,inchar,intxtp;

        # Storage for menu-id's to be erased or
        # to be displayed, initiated by the controller
ommon /blk3/iecmen,icmenu;

        # Storage for menu-id's to be erased or
        # to be displayed, initiated by a task
ommon /blk4/ietmen,itmenu;

        # Storage for operator input related messages
        # and warning messages from tasks
ogical*1 inpmsg(MSGL), iwmsg(MSGL);
ommon /blk5/inpmsg, iwmsg;

        # Storage for
        # warning messages from BG Controller,
        # read and write pointer for msgw
ogical*1 msgw(MSGWL);
ommon /blk6/msgw,msgwpr,msgwpw;

        # Menu stack, stack pointer, number of saved menu stacks
ommon /blk7/menstk(MENSTX), menstp, menstn;

        # Message stack, stack read and write pointer
logical*1 msgstk(MGSTL1);
ommon /blk8/msgstk, msgstr, msgstw;

        # Menu record number of current menu's predecessor,
        # menu record number of current or saved task menu ,
        # name of active BG Task (if any),
        # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;


        # KILL flag (1: Kill current BG Task)
        # INPUT flag (0: accept no input; 1: accept input)
ommon /blk10/killfl, inflag;

        # record numbers of initial menus known
        # to the BG Controller
        # 1)Status, 2)Select, 3)Help, 4)Control 5)Start
ommon /blk11/instat, insel, inhelp, incont, instrt;

        # number of warning messages to be displayed
        # index to warning message stack half to write into
        # first message stack record number on disc file
        # last message stack record number on disc file
ommon /blk12/msgcnt, mgstsw, mfrec, mlrec;

        # Input buffer from TT with pointer
        # flag for having done the input from operator
ommon /blk13/intt(INTTL), inttp, indone;
```

```
        # Index of permanent button (in inopt)
        # that is displayed in reverse mode
common /blk14/ipenh;

        # Saved cursor position
common /blk15/icursx, icursy;


        # logical units
common /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

        # data to communicate with the HP driver
logical*1 char(80);
integer*2 endch1,endch2,flag,count;
common /ttinbf/endch1,endch2,flag,count,char;

        # queue length for SYSF4 calls
integer*2 qlen;

integer*2 hpname(4);     # RAD50 name of device HP
data hpname/3RHP ,3R   ,3R   ,3R   /;

integer*2 area(4);       # area for itimer call
integer*2 area1(4);     # area for itimer call
        # storage for initializing commands to TT Display
logical*1 ttinit(8);
        # ESC E ESC J ESC ) B
data ttinit/07,27,69,27,74,27,41,66/;

        # storage for initializing commands to Touch System
        # A A ESC ESC Uncover-code Stop-code Control-code
        # Note -- the Control code has to be non-zero to
        #         prevent it to be regarded as trailing NUL
        #         The parity bit(odd) is set to 1
        #         The sequence is preceeded by two dummy characters
logical*1 hpinit(8);
        # A A ESC ESC ^  128
data hpinit/07,65,65,27,27,UNC,STP,CON/;

data luttd,lumsg/LUTTD,LUMSG/;
data lumstk,lumenu/LUMSTK,LUMENU/;
data lulog/LULOG/;
data lutsys/LUTSYS/;

external bgmsgm;
external bgmenm;
external bginpm;
external endfbk;




        # Call all completion routines with null input,
```

```
        # requested by RT-11, no action will be taken.
call bgmenn(NOACTN);
call bgmsgm(NOACTN);
call bginpm(NOACTN);
call endfbk(NOACTN);

        # provide sufficient queue length for
        # SYSF4 calls
qlen=32;
i=iqset(qlen);

        # Initialize common data, open files
intxtp=1;         # text pointer
msgwpr=1;         # read pointer warning msg
msgwpw=1;         # write ptr warning msg
menstp=0;         # menu stack pointer
menstn=0;         # number of menu stacks on disc
msgstr=1;         # read ptr message stack
msgstw=1;         # write ptr message stack
msgcnt=0;         # number of displayed warning messages
msgtsw=1;         # message stack half to write into
mlrec=0;          # no message stack on disc
mfrec=0;          # no message stack on disc
inttp=1;          # pointer for input buffer from TT
ipredm=0;         # no predecessor menu
icurtm=0;         # no task menu
instrt=21;        # record number of initial start menu
#++++++++++++ fill in record numbers when menufile is done
instat=27;        # initial status menu
insel=33;         # initial select menu
inhelp=39;        # initial help menu
incont=45;        # initial control menu
#+++++++++++++
ipenh=0;          # no permanent button enhanced
endch1=STP;       # stop code for input from Touch Panel
endch2=13;        # stop code for input from keyboard (CR)
count=0;          # pointer for operator input buffer

        # open all files
%      open(UNIT=lumsg,NAME='SS:MSGSTK.DAT',TYPE='UNKNOWN',
%     1ACCESS='DIRECT',FORM='UNFORMATTED',RECORDSIZE=15,
%     2MAXREC=10)
%      open(UNIT=lumstk,NAME='SS:MENSTK.DAT',TYPE='UNKNOWN',
%     1ACCESS='DIRECT',FORM='UNFORMATTED',RECORDSIZE=10,
%     2MAXREC=10)
%      open(UNIT=lumenu,NAME='SS:MENUFL.DAT',TYPE='OLD',
%     1ACCESS='DIRECT',FORM='UNFORMATTED',RECORDSIZE=136,
%     2MAXREC=500,READONLY)
%      open(UNIT=lulog,NAME='SS:LOGCMD.DAT',TYPE='UNKNOWN',
%     1ACCESS='SEQUENTIAL',FORM='FORMATTED')

        # lookup device HP on logical unit luttd
i=lookup(luttd,hpname);
```

```
        # open input from touch system
 10        read(luttd,77,END=10) endchl;
77 format(al);

        # initialize TT Display
ihp=iwritw(4,ttinit,0,luttd);

        # schedule menu mgr to display initial start menu
call itimer(0,0,0,1,area,DIMEN,bgmenm);

        # Initialize HP Touch System
ihp=iwritw(4,hpinit,0,luttd);

        # start periodic scheduling of bginpm
call itimer(0,0,0,1,areal,ACTION,bginpm);

                # get next task to call
while(1>0)
        {
        call bgciom(NOMENU);
        killfl = NOKILL;        # Reset KILL flag
#       go to (1,2), innumb;    # branch to requested task call
#       1 call taskl;
#       go to 100;
#       2 call task2;
#       100 continue;
        }
%       close(UNIT=lumsg)
%       close(UNIT=lumstk)
%       close(UNIT=lumenu)
%       close(UNIT=lulog)
end
#*******************************************************************


#*******************************************************************
#                                                                  *
#               C O M M O N   R O U T I N E S
#                                                                  *
#*******************************************************************

#       The following common routines are used by more than one
#       manager.


#*******************************************************************
#                                                                  *
#               0.1   E N D F B K
#                                                                  *
# ******************************************************************
#
# Initial release -- hn/4 Sept 79
# NAME:
#       endfbk(index)
```

```
# FUNCTION:
#       Stop any feedback initiated by an operator's input
# CALLING PROGRAM:
#       EXECUTIVE -- the very first time (requested by RT-11)
#       RT-11     -- endfbk is scheduled using itimer
# INPUT PARAMETER:
#       index -- NOACTN: dummy call from EXECUTIVE
#                index of input 'button' in array inopt
# INFORMAL INPUT:
#       inopt  -- contains input option information
# INFORMAL OUTPUT:
#       inflag -- set to ACCEPT
# NOTES:
#       This routine is a completion routine and will not be
#       interrupted by any other BG program
# ALGORITHM:
#       if call from EXECUTIVE return
#       If 'hardcopy button'   reverse button image
#       set inflag to ACCEPT
#
######################################################################


subroutine endfbk(index);

        # Menu Table declaration, number of input options
common /blk1/mentbl(TBLMAX), numopt;

        # KILL flag (1: Kill current BG Task)
        # INPUT flag (0: accept no input; 1: accept input)
common /blk10/killfl, inflag;


        # array containing all input options
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1), inopt(1,1));

        # return if call from EXECUTIVE
if(index==NOACTN) return;
        # Get action code of input
icode = inopt(CODE,index);

        # If hardcopy, reverse button
if(icode == ACODP1 | icode == ACODP2)
        call enhanc(index,REVEND);

inflag = ACCEPT;
return;
end



#*********************************************************************
```

```
#                                                                      *
#              G.2    E N H A N C
#                                                                      *
#**********************************************************************
#
# Initial release -- hn/4 Sept 79
# NAME:
#       enhanc(index,icode)
# FUNCTION:
#       Perform an enhancement function at the TT
# CALLING PROGRAM:
#       Any program of BG Controller that wants an enhancement
# INPUT PARAMETER:
#       icode -- specifies the enhancement type
#       index -- pointer to button in array inopt
# INFORMAL INPUT:
#       inopt -- input option information
# ALGORITHM:
#       Get coordinates of touch field
#       Give enhancement commands to TT
#       return
#
######################################################################

subroutine enhanc(index,icode);


        # Menu Table declaration, number of input options
common /blk1/mentbl(TBLMAX), numopt;

        # KILL flag (1: Kill current BG Task)
        # INPUT flag (0: accept no input; 1: accept input)
common /blk10/killfl, inflag;

common /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

        # Array containing all input option information
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1), inopt(1,1));

        # arrays containing the command strings to do
        # the requested enhancement
logical*1 icmd1(22), icmd2(12), icmd3(6);
        # ESC3 ESC&axxrxxC ESC1 ESC&axxC ESC1 -- set tabs
data icmd1/07,27,51,27,38,97,00,00,114,00,00,67,27,49,
27,38,97,00,00,67,27,49/;

        # ESC&dxx ESCI ESC&d@ -- one line of enhancement
data icmd2/07,36,27,38,100,00,27,73,27,38,100,64/;

        # ESCi ESCB -- back tab, down one row
data icmd3/07,36,27,105,27,66/;
```

```
        # set enhancement typ
if(icode == REV)          # inverse video
   icmd2(6)=66
else if(icode == REVEND)          # inverse,half bright
   icmd2(6)=74
else if(icode == BLKBEG)          # blinking, half bright
   icmd2(6)=73
else if(icode == BLKEND)          # inverse, half bright
   icmd2(6)=74;

        # set coordinates of button into command string
ilx=inopt(LX,index);     # left x
irx=inopt(RX,index);     # right x
iuy=inopt(UY,index);     # upper y
ily=inopt(LY,index);     # lower y

icmd1(7)=iuy/10+48;
icmd1(8)=mod(iuy,10)+48;
icmd1(10)=irx/10+48;
icmd1(11)=mod(irx,10)+48;
icmd1(18)=ilx/10+48;
icmd1(19)=mod(ilx,10)+48;

ihp=iwritw(11,icmd1,0,luttd);
ihp=iwritw(6,icmd2,0,luttd);
while(iuy<ily)  # while more lines to enhance
  {
  iuy=iuy+1;
  ihp=iwritw(3,icmd3,0,luttd);
  ihp=iwritw(6,icmd2,0,luttd);
  }


return;
end




#*********************************************************************
#                                                                   *
#              0.3     E R R M S G
#                                                                   *
#*********************************************************************
#
# Initial release -- hn/7 Sept 79
# NAME:
#       errmsg(mtyp,mlen,mbuff)
# FUNCTION:
#       store a message (input related or warning) in the temporary
#       message stack msgw or inpmsg and store an end-of-message
#       word
# CALLING PROGRAM:
```

```
#       Any program of the BG Controller that wants a message
#        to be displayed
# INPUT PARAMETER:
#       mtyp    - typ of message: INM - input related message
#                                 WM  - warning message
#       mlen    - length of message (words)
#       mbuff   - array containing ASCII text
# INFORMAL INPUT:
#       msgw    - temp. storage for warning messages
#       msgwpr  - read pointer for msgw
#       msgwpw  - write pointer for msgw
# INFORMAL OUTPUT:
#       Same as informal input
#       Additionally
#       inpmsg  - text of input related message
# NOTE:
#       An overflow of this temporary stack will not happen, its
#        length is sufficient for all messages generated by any
#        completion level execution
# ALGORITHM:
#       while more literals to store
#          copy a word of literals
#          increment the write pointer
#          if the write pointer > stack-length
#             set write pointer = 1
#       write an end-of-message word
#       increment write pointer
#       if write pointer > stack-length
#          set write pointer = 1
#
######################################################################

subroutine errmsg(mtyp,mlen,mbuff);



        # Storage for operator input related messages
        # and warning messages from tasks
logical*1 inpmsg(MSGL), iwmsg(MSGL);
common /blk5/inpmsg, iwmsg;

        # Storage for
        # warning messages from BG Controller,
        # read and write pointer for msgw
logical*1 msgw(MSGWL);
common /blk6/msgw,msgwpr,msgwpw;

logical*1 mbuff(MSGL);

        # Store input related message
if(mtyp == INM)
   {
   for(i=1;i<=mlen;i=i+1)
      inpmsg(i) = mbuff(i);     # copy text
```

```
  for(j=mlen+1; j<MSGL; j=j+1)
     inpmsg(j) = '  '; # blank into rest
  }

       # store warning message
lse
  {
  for(i=1;i<=mlen;i=i+1)
     {
     msgw(msgwpw) = mbuff(i);  # copy a word
       # increment write pointer
     if(msgwpw >= MSGWL) msgwpw = 1
     else msgwpw = msgwpw + 1;
     }
       # write end-of-message word, increment pointer
  msgw(msgwpw) = ENDOFM;
  if(msgwpw >= MSGWL) msgwpw = 1
  else msgwpw = msgwpw + 1;
  }

eturn;
nd


#**********************************************************************
#
#               0.4   A S C I N T
#
#**********************************************************************
#
# Initial release -- hn/14 november 79
# NAME:
#       ascint(p1,p2,p3)
# FUNCTION:
#       convert an ASCII coded integer into an integer
# ALGORITHM:
#       p3=(p1-40)*30 + (p2-40)
#
######################################################################

subroutine ascint(p1,p2,p3);
integer*2 p1,p2,p3;

p3=(p1-40)*30 + p2-40;

return;
end
```

```
#**********************************************************************
#                                                                     *
#             1.  C O N T R O L   I/O   M A N A G E R
#                                                                     *
#**********************************************************************
#
# initial release -- hn/24 aug 79
# NAME:
#      bgciom(iflag)
# FUNCTION:
#      process an operator input request
# CALLING PROGRAMS:
#      - any BG Task
#      - EXECUTIVE
# INPUT PARAMETER:
#      iflag -- 0: EXECUTIVE expects id of next task [NOMENU]
#               1: A BG Task requests a menu to be displayed
#                  (Menu record number in itmenu)  [MENU]
# INFORMAL INPUT:
#      itmenu -- record number of any task menu to be
#                          displayed
# INFORMAL OUTPUT:
#      intext(TEXTL) -- one line of text  or
#      intxtp     -- pointer for text buffer
#      innumb     -- one integer digit or
#      inchar     -- one ASCII character
# ALGORITHM:
#      bgciom(iflag):
#      clear input storage locations
#      if call from a task
#         schedule the MENU MGR to display menu immediately
#      loop until input ready
#      return input to caller
######################################################################


subroutine bgciom(iflag)

external bgmenm


       # Operator input storage declarations
common /blk2/intext(TEXTL),innumb,inchar, intxtp;

       # indone, flag input done
common /blk13/intt(INTTL), inttp, indone;

integer*2 area(4);   # area for itimer call

               # Preset input storage locations
innumb = -1;
inchar = '##';
intext(1)='##';
```

```
for (i=2;i<=TEXTL;i=i+1)
 intext(i) = '  ';
       # preset pointer
intxtp = 1;

       # Schedule MENU MGR immediately after 1 clock tick
       # if task menu is to be displayed
if(iflag == MENU)
 call itimer(0,0,0,1,area,DTMEN,bgmenn);


               # Loop until an input is done
       # preset flag
indone = NODONE;
while(indone == NODONE);
return;
end;

#**********************************************************************
```

```
#**********************************************************************
#                                                                     *
#          2. I N P U T   M A N A G E R
#                                                                     *
#**********************************************************************
#
# initial release -- hn/24 aug 79
# NAME:
#       bginpm(inp)
# FUNCTION:
#       Processing of operator inputs from touch panel/keyboard
# CALLING PROGRAM:
#       EXECUTIVE -- the very first time (requested by RT-11)
#       RT-11 -- bginpm is scheduled by an input interrupt driver
# INPUT PARAMETER:
#       inp -- 0 (NOACTN): Dummy call from EXECUTIVE
#              TTINP:  Input from TT
#              KEYINP: Input from keyboard
# INFORMAL INPUT:
#       inopt -- contains input options and action codes
#       intt  -- input buffer
# INFORMAL OUTPUT:
#       intext(TEXTL) -- one line of text or
#       innumb     -- one integer number or
#       inchar     -- one ASCII character
#       inflag     -- ACCEPT - accept input
#                     NOACPT - do not accept input
# NOTES:
#       This routine is a completion routine and will not be
#       interrupted by any other BG program.
#       Same applies to all routines called from bginpm.
# ALGORITHM:
#       re-schedule its call (after .1 seconds)
#       convert and check input
#       if input invalid
#           schedule MESSAGE MGR for 'Illegal Input' and return
#         else
#           schedule MSG MGR to erase any input related message
#       provide feedback to operator (reverse button)
#       call LOG MGR to log command
#       if a new menu is to be displayed
#           schedule MENU MGR to display new menu
#         else
#           schedule MENU MGR to erase current menu in 1 second
#       if input for task
#           save input for CONTROL I/O MGR
#         else
#           call the CONTROL COMMAND MGR
#       return
#
######################################################################


subroutine bginpm(inp)
```

```
xternal bgmsgm;
xternal bgmenm;
xternal bginpm;


        # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;

        # Operator input storage declarations
ommon /blk2/intext(TEXTL),innumb,inchar,intxtp;

        # Storage for menu-id's to be erased or
        # to be displayed, initiated by the controller
ommon /blk3/iecmen,icmenu;

        # Storage for menu-id's to be erased or
        # to be displayed, initiated by a task
ommon /blk4/ietmen,itmenu;

        # Storage for operator input related messages
        # and warning messages from tasks
        # Menu stack, stack pointer, number of saved menu stacks
ommon /blk7/menstk(MENSTX), menstp, menstn;


        # Menu record number of current menu's predecessor,
        # menu record number of current or saved task menu ,
        # name of active BG Task (if any),
        # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;


        # KILL flag (1: Kill current BG Task)
        # INPUT flag (0: accept no input; 1: accept input)
ommon /blk10/killfl, inflag;

        # input buffer, pointer, input-done flag
ommon /blk13/intt(INTTL), inttp, indone;

ommon /ttinbf/endch1,endch2,flag,count,char;
ogical*1 char(80);
nteger*2 endch1,endch2,flag,count;


nteger*2 inx,iny

        # Storage for keyboard entry
nteger*2 inkey

nteger*2 area1(4);      # area for itimer to
                        # schedule the MENU MGR
nteger*2 area2(4);      # area for itimer to
                        # schedule the MESSAGE MGR
```

```
nteger*2 area3(4); # area for itimer to
                   # schedule itself

        # Array containing all input options
nteger*2 inopt(OPTL,INMAX);
quivalence (mentbl(1), inopt(1,1));

        # Do not process first dummy call from EXECUTIVE
f(inp == NOACTN) return;
        # Re-schedule itself
all itimer(0,0,0,6,area3,ACTION,bginpm);

        # return if no input done
f(count==0) return;
        # reset buffer pointer
ount=0;
        # Do not accept any input if inflag set
f(inflag == NOACPT)  return;


 inx = char(2);
 iny = char(3);
 j = 1;
invflg=0;       # flag set if valid coordinates found
 while(j<=numopt)
 {
 i=j;  # save index
 if(inx<=inopt(RX,j) & inx>=inopt(LX,j)
  & iny<=inopt(LY,j) & iny>=inopt(UY,j))
        { invflg=1;
          break; }
 j=j+1;
 }

if(invflg==0)      # if invalid input,schedule MSG MGR
 {
 call errmsg(INM,13,'INVALID INPUT');
 call itimer(0,0,0,1,area2,DINPUT,bgmsgm);
 return;
 }


call bglogm(i); # log command
call feedbk(i); # provide feedback

                # interpret input action code
icode = inopt(CODE,i);
if(icode >= ACODP1 & icode<=ACODCN)      # processed by bgcmdm
  call bgcmdm(i);              # pass input
return;
if(icode == ACODEM)            # display new menu
 {
 icmenu = inopt(ANNEX,i);
 call itimer(0,0,0,1,areal,mentyp,bgmenm); # schedule MENU MGR
```

```
  }

lse    # erase current display in 1 second
 {
 if(mentyp == TASK)     # current task menu
   {
   ietmen = icurtm;     # task menu id
   call itimer(0,0,1,0,areal,ETMEN,bgmenm);
   }
 else if(menstp>0)
   {
   iecmen = menstk(menstp);
       # Mark menu on top of stack to be erased
   menstk(menstp) = -menstk(menstp);
   call itimer(0,0,1,0,areal,ECMEN,bgmenm); # schedule MENU MGR
   }
 }

f(icode == ACODET)             # if text input
 {
 intext(intxtp) = inopt(ANNEX,i);
 intxtp = intxtp+1;
 if(++++++ c/r lf or so)
   indone = DONE;
 return;
 }

if(icode == ACODEN)     # if integer
 {
 innumb = inopt(ANNEX,i);
 indone = DONE;
 return;
 }

if(icode == ACODEC)     # if character
 {
 inchar = inopt(ANNEX,i);
 indone = DONE;
 return;
 }

end



#*********************************************************************
#                                                                   *
#               2.1   F E E D B K
#                                                                   *
#*********************************************************************
#
# Initial release -- hn/4 Sept 79
# NAME:
```

```
       feedbk(index)
 FUNCTION:
       Provide feedback to an operator's input
 CALLING PROGRAM:
       bginpm
 INPUT PARAMETER:
       index -- index for input in array inopt
 INFORMAL INPUT:
       inopt -- array containing input information
 INFORMAL OUTPUT:
       inflag -- NOACPT - Do not accept any input
 ALGORITHM:
       Provide accustic feedback
       Set inflag to NOACPT
       If(STATUS|SELECT|HELP|CONTROL|ACKNOWLEDGE)
                return;
       else
         reverse button
         schedule endfbk in .5 seconds to reverse button back

###################################################################

subroutine feedbk(index);

external endfbk;


       # Menu Table declaration, number of input options
common /blk1/mentbl(TBLMAX), nunopt;


       # KILL flag (1: Kill current BG Task)
       # INPUT flag (0: accept no input; 1: accept input)
common /blk10/killfl, inflag;

       # Array containing input information
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1), inopt(1,1));

       # array for itimer call
integer*2 area(4);

       # get action code of input
icode = inopt(CODE,index);

#++++++ provide accustic feedback

       # Do not accept input during feedback period
inflag = NOACPT;

       # If Status,Select,Help,Control,Acknowledge: return
       # else reverse button, schedule end of feedback
if(icode < ACODAC)
  {
```

```
call enhanc(index,REV);
call itimer(0,0,0,30,area,index,endfbk);
}
eturn;
nd
```

```
*******************************************************************
                                                                  *
         3. C O N T R O L   C O M M A N D   M A N A G E R       *
                                                                  *
*******************************************************************

Initial release -- hn/24 aug 79
NAME:
      bgcmdm(inx)
FUNCTION:
      Process operator inputs from permanent displayed
        control 'buttons' and from menus initiated by
        the BG Controller.
CALLING PROGRAM:
      Input Manager (bginpm)
INPUT PARAMETER:
      inx - index in inopt, specifies kind of input
INFORMAL INPUT:
      mentbl -- menu table with all input options
      mentyp -- type of current menu
INFORMAL OUTPUT:
      killfl -- 1: current task is to be killed
ALGORITHM:
      If input from permanent button
        { If PRINT CRT1/2: Initiate hardcopy;
          If ACKNOWLEDGE : Schedule MSG MGR to erase warning msg
          If STATUS,HELP,
           SELECT,CONTROL:
           { If control function is active, erase menu;
             If function not active, display its first menu;
           }
        }
      If input from a controller menu button
        { If SELECT-menu (start a new task)
             { Erase SELECT menu;
               Set the KILL flag
               Return new task-id;
              }
          If CONTROL-menu:
            Call the PB MGR to process the input;
        }
###################################################################


ubroutine bgcmdm(inx);

xternal bgmsgm;
xternal bgmenm;


      # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), nunopt;

      # Operator input storage declarations
```

```
ommon /blk2/intext(TEXTL),innumb,inchar,intxtp;

        # Storage for menu-id's to be erased or
        # to be displayed, initiated by the controller
ommon /blk3/iecmen,icmenu;

        # Storage for menu-id's to be erased or
        # to be displayed, initiated by a task
ommon /blk4/ietmen,itmenu;


        # Menu stack, stack pointer, number of saved menu stacks
ommon /blk7/menstk(MENSTX), menstp, menstn;


        # Menu record number of current menu's predecessor,
        # menu record number of current or saved task menu ,
        # name of active BG Task (if any),
        # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;


        # KILL flag (1: Kill current BG Task)
        # INPUT flag (0: accept no input; 1: accept input)
ommon /blk10/killfl, inflag;

        # record numbers of initial menus known
        # to the BG Controller
        # 1)Status, 2)Select, 3)Help, 4)Control 5)Start
ommon /blk11/instat, insel, inhelp, incont, instrt;

        # arrays for itimer calls
nteger*2 area1(4);
nteger*2 area2(4);

        # array for input option
nteger*2 inopt(OPTL,INMAX);
quivalence (mentbl(1), inopt(1,1));


        # save action code
code = inopt(CODE,inx);

f(inx <= CONMAX)       # permanent control buttons
 {if(icode == ACODP1)          # hardcopy CRT1
     {
     #++++++++++ initiate hardcopy
     return;}

  else if(icode == ACODP2)     # hardcopy CRT2
     {
     #+++++++++ initiate hardcopy
     return;}
```

```
 else if(icode == ACODAC)      # ACKNOWLEDGE
    {
    # Erase warning message
    call itimer(0,0,0,1,area1,EWARN,bgmsgm);
    return;}

 else # STATUS,SELECT,HELP,CONTROL: either
      # erase current menu or display initial one
    {
    kode = icode-8;    # kode and mentyp same meaning
    if(mentyp == kode)         # erase menu, switch off function
      {
      iecmen = menstk(menstp);         # copy current menu-id
          # Mark menu on top of stack
      menstk(menstp) = -menstk(menstp);
      call itimer(0,0,0,1,area2,ECMEN,bgmenm);
      return;
      }
    else       # display initial menu, switch on
      {
      if(icode == ACODST)      # status
        icmenu = instat
      else if(icode == ACODSE) select
        icmenu = insel
      else if(icode == ACODHP)         # help
        icmenu = inhelp
      else if(icode == ACODCN)         # control
        icmenu = incont;
      call itimer(0,0,0,1,area2,kode,bgmenm);
      return;
      }
    }
 }

      # input from a controller menu
lse if(icode == ACODTK)        # new task
 {
 innumb = inopt(ANNEX,inx);    # task-id for EXECUTIVE
 killfl = KILL;                # kill any running task
 ietmen = icurtm;
 call itimer(0,0,0,1,area1,ETMEN,bgmenm); # erase task menu

      # erase current select menu
 iecmen = menstk(menstp);
 menstk(menstp) = -menstk(menstp);
 call itimer(0,0,0,1,area2,ECMEN,bgmenm);
 return;
 }

lse    # input for PB MGR
 {
 call bgpbm(inx);
 return;
 }
```

```
eturn;
nd;

************************************************************************
```

```
*******************************************************************
                                                                  *
         4. M E N U   M A N A G E R
                                                                  *
*******************************************************************

Initial release -- hn/6 Sept 79
NAME:
      bgmenm(itype)
CALLING PROGRAM:
      EXECUTIVE -- the very first time (requested by RT-11)
      RT-11     -- bgmenm is to be scheduled using itimer call
FUNCTION:
      Update the enhancements of some permanent 'buttons'
      Process other program's request to display/erase a menu
      Read a menu from the menu file and update the TT display
      Do not allow any operator input while changing the display
INPUT PARAMETER:
      itype -- specifies the kind of action requested by the
               scheduling program
        NOACTN - dummy call from EXECUTIVE
        DIMEN  - display initial start menu
        DSMEN  - display status menu
        DSEMN  - display select menu
        DHMEN  - display help menu
        DCMEN  - display control menu
        DTMEN  - display BG Task menu
        ECMEN  - erase a control menu
        ETMEN  - erase a task menu
INFORMAL INPUT:
      menstk - menu stack
      menstp - menu stack pointer
      icmenu - controller menu to be displayed
      iecmen - controller menu to be erased
      itmenu - task menu to be displayed
      ietmen - task menu to be erased
      mentyp - type of current menu
      mentbl - menu table for storage of input
               and image information
INFORMAL OUTPUT:
      inflag -- ACCEPT: accept operator input
NOTE:
      This routine is a completion routine and will not be
      interrupted by any other BG program
ALGORITHM:
      if NOACTN  return;
      set inflag to NOACPT
      if DIMEN
         Display any next menu
      if DSMEN,DSEMN,DHMEN,DCMEN
         if displayed menu is of the same type
              replace the menu by the new one
         else save the old one and display the new one
      if DTMEN
```

```
         Save new task menu id
         if a task menu is being displayed
            replace it by the new one
      if ECMEN
         if the menu is being displayed
            display the next saved menu
      if ETMEN
         if the task menu is being displayed
            display any waiting menu
      set inflag to ACCEPT
      return

################################################################################


ıbroutine bgmenm(itype);


      # Menu Table declaration, number of input options
ɔmmon /blk1/mentbl(TBLMAX), numopt;


      # Storage for menu-id's to be erased or
      # to be displayed, initiated by the controller
ommon /blk3/iecmen,icmenu;

      # Storage for menu-id's to be erased or
      # to be displayed, initiated by a task
ommon /blk4/ietmen,itmenu;


      # Menu stack, stack pointer, number of saved menu stacks
ommon /blk7/menstk(MENSTX), menstp, menstn;


      # Menu record number of current menu's predecessor,
      # menu record number of current or saved task menu ,
      # name of active BG Task (if any),
      # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;


      # KILL flag (1: Kill current BG Task)
      # INPUT flag (0: accept no input; 1: accept input)
ommon /blk10/killfl, inflag;

ommon /blk11/instat,insel,inhelp,incont,instrt;
      # Array containing all input options
nteger*2 inopt(OPTL,INMAX);
quivalence (mentbl(1), inopt(1,1));

      # No action on dummy call from EXECUTIVE
f(itype == NOACTN) return;
```

```
      # Do not allow any operator input
ıflag = NOACPT

      # Display initial start menu
:(itype == DIMEN)
 {icmenu = 0;          # reset store
  call mdispl;         # display any waiting menu or initial one
 }

      # Display Status,Select,Help,Control menu
lse if(itype>=DSMEN & itype<=DCMEN)
 {
 if(itype == mentyp)  # menu of same type on TT
    {
    call mepop;                # erase current menu
    if(mepush(1) == OK)        # push new menu
       call mdispl;    # display new menu
    }
 else                  # menu of different type on TT
    {
    if(mepush(1) == OK)        # Menu pushed on top of menu-stack
       call mdispl;    # Display the new menu
    }
 icmenu = 0;           # reset store
 }

      # Display task menu
lse if(itype == DTMEN)
{
if(icurtm != itmenu)   # if different task menu
 {
 icurtm = itmenu;      # save menu id
 if(mentyp==TASK | mentyp==START)# task menu is being displayed
                                 # or initial start menu
    call mdispl;       # display new task menu
 }
itmenu = 0;            # reset store
}

      # Erase controller menu
lse if(itype == ECMEN)
 {
      # get absolute menu id on top of menu stack
 i = menstk(menstp);
 if(i<0) i=-i;
 if(iecmen == i)       # menu is being displayed
    {
    call mepop;        # pop menu on top of stack
    call mdispl;       # display next menu
    }
 iecmen = 0;           # reset store
 }

      # Erase task menu
```

```
lse if(itype == ETMEN)
  {
  if(icurtm == ietmen) # menu to be erased is saved or
                       # being displayed
     {
     icurtm = 0;       # erase menu store
     if(mentyp == TASK)  # old task menu being displayed
        call mdispl;   # display new task menu
     }
  ietmen = 0;          # reset store
  }

      # Allow operator input
nflag = ACCEPT;

eturn;
nd
```

```
*******************************************************************
                                                                  *
              4.1    M E P U S H
                                                                  *
*******************************************************************

 Initial release -- hn/6 Sept 79
 NAME:
      mepush(i)
 CALLING PROGRAM:
      bgmenm
 FUNCTION:
      Push a new controller menu id on top of the menu stack,
       save any full stacks on disc
 INPUT PARAMETER:
      i -- dummy parameter (needed for function call)
 INFORMAL INPUT:
      menstk - menu stack
      menstp - menu stack pointer
      menstn - number of saved stacks on disc
      icmenu - menu-id to push
 FUNCTION OUTPUT:
      OK    - new menu-id on top of menu stack
      NOTOK - a full stack could not be written onto disc
               new menu-id is not on top of menu
 INFORMAL OUTPUT:
      menstp - menu stack pointer
 ALGORITHM:
      if menu-stack full
         increment stack counter
         write stack onto disc
         if write not possible
            decrement stack counter
```

```
            initiate display of a warning message
            return NOTOK
         else reset stack pointer
      else increment stack pointer
      store menu-id on stack
      return OK

###################################################################


inction mepush(i);

xternal bgmsgm



      # Storage for menu-id's to be erased or
      # to be displayed, initiated by the controller
ommon /blk3/iecmen,icmenu;


      # Menu stack, stack pointer, number of saved menu stacks
ommon /blk7/menstk(MENSTX), menstp, menstn;

      # Array for itimer call
nteger*2 area(4);
nteger*2 lumstk;
ata lumstk/LUMSTK/;


f(menstp>=MENSTX)       # menu stack full
  {
  menstn = menstn+1;    # incr counter for saved stacks

      # write stack onto disc, if error goto 9999
     write(lumstk'menstn,ERR=9999) menstk
  menstp = 1;           # reset stack pointer
  }
lse              # no stack overflow
  menstp = menstp+1;    # incr stack pointer

      # Store menu-id on stack
enstk(menstp) = icmenu;
epush = OK;
eturn;

      # If write could not be done properly
999   menstn = menstn-1;       # reset counter

      # Initiate display of warning message
  call errmsg(WM,27,'MENU STACK WRITE IMPOSSIBLE');
  call itimer(0,0,0,1,area,DWARN,bgmsgm);
  mepush = NOTOK;
eturn;
```

```
nd


*********************************************************************
                                                                    *
              4.2    M E P O P
                                                                    *
*********************************************************************

 Initial release -- hn/6 Sept 79
 NAME:
       mepop
 FUNCTION:
       Pop the menu on top of the stack,
       read any menu stacks saved on disc
 INFORMAL INPUT:
       menstk - menu stack
       menstp - menu stack pointer
       menstn - number of saved stacks on disc
 INFORMAL OUTPUT:
       menstp - menu stack pointer
 ALGORITHM:
       decrement stack pointer
       while(stack in memory or on disc)
          while (stack pointer > 0)
             if(menu-id positiv) return
             else decrement stack pointer
          if(stack saved on disc)
             read stack
             if(improper read)
                initiate warning message
                return
             decrement stack counter
             reset stack pointer to top of stack
       return

#####################################################################


ubroutine mepop;

xternal bgmsgm;



       # Menu stack, stack pointer, number of saved menu stacks
common /blk7/menstk(MENSTX), menstp, menstn;

common /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

       # array for itimer call
integer*2 area(4);
```

```
      # decrement stack pointer
;(menstp>0) menstp = menstp-1;

      # while there is a stack in memory or on disc
ile(menstp>0 | menstn>0)
 {
         # while a stack in memory
 while(menstp>0)
    {
    if(menstk(menstp)>0) return       # next menu found
    else menstp = menstp-1;
    }

      # if stack saved on disc
 if(menstn>0)
    {
      read(lumstk'menstn,ERR=9999) menstk
    menstn = menstn-1;          # update stack counter
    menstp = MENSTX;            # update stack pointer
    }
 }
eturn; # return, stack is empty

      # If read could not be done properly
      # Initiate display of warning message
999  call errmsg(WM,26,'MENU STACK READ IMPOSSIBLE');
    call itimer(0,0,0,1,area,DWARN,bgmsgm);
eturn;
nd
```

```
*********************************************************************
                                                                    *
              4.3   M D I S P L
                                                                    *
*********************************************************************
```

```
 Initial release -- hn/7 Sept 79
 NAME:
      mdispl
 CALLING PROGRAM:
      bgmenm
 FUNCTION:
      Get the next menu to be displayed
        a) a controller menu on stack
        b) a task menu
        c) initial start menu
      Read and display the menu image
      Read and store the input option information
      Update the enhancement of certain permanent buttons
      Generate error messages if read/write impossible
 INFORMAL INPUT:
```

```
#      menstk - controller menu stack
#      icurtm - task menu
#      instrt - initial start menu
#      menstp - menu stack pointer
#INFORMAL OUTPUT:
#      inopt  - array containing input option information
#ALGORITHM:
#      If controller menu on stack
#        get menu-id
#      else if task menu
#        get menu-id
#      else get initial start menu-id
#
#      while(image records on disc)
#         read a record
#         if read error, display message,  return
#         display image of the record
#         if write error, display message,  return
       read all input option records
       if read eror, display message, return
       update enhancement of certain permanent buttons
       return

##################################################################

ubroutine mdispl;

xternal bgmsgm;

       # Menu stack, stack pointer, number of saved menu stacks
ommon /blk7/menstk(MENSTX), menstp, menstn;


       # Menu record number of current menu's predecessor,
       # menu record number of current or saved task menu ,
       # name of active BG Task (if any),
       # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;


ommon /blk11/instat,insel,inhelp,incont,instrt;

       # number of warning messages to be displayed
       # index to warning message stack half to write into
       # first message stack record number on disc file
       # last message stack record number on disc file
ommon /blk12/msgcnt, mgstsw, mfrec, mlrec;


       # array for itimer call
nteger*2 area(4);


       # get menu-id to be displayed
```

```
i(menstp > 0)          #  controller-menu
  irec = menstk(menstp)
ese if(icurtm != 0)    # task menu
  irec = icurtm
ese irec = instrt;     # Initial start menu

        # reset offset for disc records
i= 0;

        # preset flag
one = NODONE;

        # while any image records on disc, read and display them
ile(idone != DONE &  i<=3)
  {
        # read one image record
  call rimage(irec+i,iflag,idone);

        # if read error, display message and return
  if(iflag == NOTOK)
     {
     call errmsg(WM,20,'MENU READ IMPOSSIBLE');
     call itimer(0,0,0,1,area,DWARN,bgmsgm);
     return;
     }
        # increment record offset
  i = i+1;

        # display one image record
  call dimage(iflag);

        # if write error, display message and return
  if(iflag == NOTOK)
     {
     call errmsg(WM,23,'MENU DISPLAY IMPOSSIBLE');
     call itimer(0,0,0,1,area,DWARN,bgmsgm);
     return;
     }
  }

        # read all input option information
all rinopt(irec,iflag);
        # if read error, display message and return
(iflag == NOTOK)
  {
  call errmsg(WM,20,'MENU READ IMPOSSIBLE');
  call itimer(0,0,0,1,area,DWARN,bgmsgm);
  return;
  }
        # Update any enhancements of control buttons
all updenh;
eturn;
nd
```

```
#*****************************************************************
#                                                                *
#              4.3.1     R I M A G E
#                                                                *
#*****************************************************************
#
#Initial release -- hn/7 Sept 79
#NAME:
#      rimage(menrec,iflag,idone)
#CALLING PROGRAM:
#      mdispl
#FUNCTION:
#      read a specified menu-record into mentbl
#      if end-of-image , set flag idone
#INPUT PARAMETER:
#      menrec - record number to read
#OUTPUT PARAMETER:
#      iflag - NOTOK: Read error
#              OK   : read ok
#      idone - DONE  : end-of-image recognized
#              NODONE: no end-of-image
#INFORMAL OUTPUT:
#      mentbl - menu image starting at mentbl(IMAGEX)
#ALGORITHM:
#      read image record
#      if read ok
#         if last image record
#            set flag to DONE
#         return flag OK
#      else return flag NOTOK

#################################################################

ubroutine rimage(menrec,iflag,idone);

       # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;

ommon /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

       # array for temporary image storage
ogical*1 image(TBLMAX);
quivalence (mentbl(IMAGEX), image(1));

rec=menrec;
       # read image record
       read(lumenu'irec,ERR=9999) (image(i),i=1,540)

       # if last record read, set done flag
f(image(540) == ENDOFR)        # last word in record = code
  idone = DONE;
```

```
      # return read ok
ilag = OK;
rturn;

      # return NOTOK, if read error
99 iflag = NOTOK;
rturn;
ed


*********************************************************************
                                                                    *
             4.3.2      D I M A G E
                                                                    *
*********************************************************************

Initial release -- hn/10 Sept 79
NAME:
      dimage(iflag)
CALLING PROGRAM:
      mdispl
FUNCTION:
      Display the menu image stored in image
OUTPUT PARAMETER:
      iflag - OK: No write error
              NOTOK: Write error
INFORMAL INPUT:
      image  - image of menu display
               image starts at mentbl(IMAGEX)
ALGORITHM:
      while there are more image commands
         write commands to display
         if write error, return NOTOK
      return OK

####################################################################

ubroutine dimage(iflag);

      # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;

ommon /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

      # output buffer with pointer
ogical*1 buff(62);
nteger*2 buffx;

      # array containing the menu image
ogical*1 image(TBLMAX);
quivalence (mentbl(IMAGEX), image(1));

      # command string to position cursor to start
      # of input related message field after the new
```

```
      # image has been transferred to the TT Display
lgical*1 cmd(10);
cta cmd/07,27,38,97,50,49,114,50,49,67/;


      # preset ending index used in loop
idend = 0;
      # send always '$' at beginning of command string
ff(1)=36; buff(2)=36;

      # do forever, check if more commands stored
      # if any and less than display buffer size,
      # write the commands
ile(1>0)
 {
      # preset begin index and end index for checks
 indbeg = indend+1;
 indend = indend+60;
 buffx=3;     # first entry for command string

      # check all entries in image between indbeg and indend
 for(index=indbeg; index<=indend; index=index+1)
    {
    buff(buffx)=image(index);
    buffx=buffx+1;
      # if end-of-image or end-of-physical-record
      # write rest of image
    if(index>TBLMAX | image(index)==ENDOFI)
      {
      buffx=buffx-1;
      buff(buffx)=36; # '$'
      nwords=buffx/2;
      ihp=iwritw(nwords,buff,0,luttd);
         # return flag write OK

      # position cursor after image commands done
      ihp=iwritw(5,cmd,0,luttd);
      iflag = OK;
      return;
      }
    }

      # write image
  ihp=iwritw(31,buff,0,luttd);
 }


000 format(80a1);

      # write error if here
999 iflag = NOTOK;
eturn;
nd
```

```
#*********************************************************************
#                                                                    *
#             4.3.3    R I N O P T
#                                                                    *
#*********************************************************************
#
#Initial release --hn/10 Sept 79
#NAME:
#      rinopt(menrec,iflag)
#CALLING PROGRAM:
#      mdispl
#FUNCTION:
#      read all input option information
#      read menu parameter and save them
#INPUT PARAMETER:
#      menrec - number of first record of the menu
 OUTPUT PARAMETER:
       iflag  - OK: No read error
                NOTOK: read error
 INFORMAL OUTPUT:
       mentbl - input option information
       numopt - number of input options
       ipredm - record number of predecessor menu
       mentyp - type of this menu
 ALGORITHM:
       set initial index in mentbl for read
       while there are any records to read
          read record
          if error, return NOTOK
          if last record, break
       store parameter
       return OK

#####################################################################

ubroutine rinopt(menrec,iflag);

       # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;


       # Menu record number of current menu's predecessor,
       # menu record number of current or saved task menu ,
       # name of active BG Task (if any),
       # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;



       # record numbers of initial menus known
       # to the BG Controller
```

```
      # 1)Status, 2)Select, 3)Help, 4)Control 5)Start
cmmon /blk11/instat, insel, inhelp, incont, instrt;

c mon /blk17/lumsg,lunstk,lumenu,luttd,lutsys,lulog;

      # array to store input option information
iteger*2 inopt(OPTL,INMAX);
euivalence (mentbl(1),inopt(1,1));

iteger*2 p1,p2,p3;


      # Compute initial read index in mentbl
i(menrec == instrt)    # if initial start menu
                       # with control button info
  index=1
 lse index=OPTL*CONMAX+1;       # first entry after control button

      # first record number after last one for input
j= menrec+IMGREC+OPTREC;

 index+MRL-1;  # last index for read into mentbl
 ec=menrec+IMGREC;      # first record with input options
     read(lumenu'irec,ERR=9999) (mentbl(ind),ind=index,k)
 dex=index+MRL;
 MRL*2;                 # last entry in mentbl
 ec=menrec+IMGREC+1;    # next record
     read(lumenu'irec,ERR=9999) (mentbl(ind),ind=index,k)

      # all information is read, save menu parameters
3=inopt(1,36);
1=p3/256; p2=mod(p3,256); call ascint(p1,p2,p3);
predm=p3;       # menu id of predecessor menu

3=inopt(2,36);
1=p3/256; p2=mod(p3,256); call ascint(p1,p2,p3);
entyp=p3;       # menu type

3=inopt(3,36);
1=p3/256; p2=mod(p3,256); call ascint(p1,p2,p3);
unopt=p3;       # number of input options
f(mentyp!=START)        # not initial start menu
 nunopt=nunopt+CONMAX;  # add permanent buttons

      # convert all ASCII coded integer into integer
      # (coordinates,action codes, annex if necessary)
f(menrec == instrt)    # start menu
  istart=1
lse istart=CONMAX+1;

end=nunopt;

or(i=istart; i<=iend; i=i+1)  # for all inp options
 {
```

```
for(k=7; k<=11; k=k+1)        # for all coord & act codes
  {
  p3=inopt(k,i); p1=p3/256; p2=mod(p3,256);
  call ascint(p1,p2,p3);
  inopt(k,i)=p3;
  }
if(p3==ACODEM | p3==ACODEN |p3==ACODTK)  # convert annex
  {
  p3=inopt(ANNEX,i); p1=p3/256; p2=mod(p3,256);
  call ascint(p1,p2,p3);
  inopt(ANNEX,i)=p3;
  }
}


      # return OK
lag = OK;
turn;

      # if here, process read error
99 iflag = NOTOK;
turn;
d
```

```
*********************************************************************
                                                                    *
              4.3.4      U P D E N H
                                                                    *
*********************************************************************
```

```
Initial release -- hn/11 Sept 79
NAME:
      updenh
CALLING PROGRAM:
      mdispl
FUNCTION:
      Update the enhancement of certain permanent
        control buttons
INFORMAL INPUT:
      ipenh  -- specifies any permanent button being reversed
                 ( index of button input in inopt)
      mentyp -- typ of new menu
INFORMAL OUTPUT:
      ipenh  -- 0: new menu is a task menu
                x: specifies the permanent button that is reversed
                   (index in inopt)
ALGORITHM:
      if there is an enhancement
          if old menu-typ = new menu-typ , return
          else reverse old enhancement
```

```
#       if no new enhancement is to be made , return
#       else reverse new enhancement
#       return

##################################################################

ubroutine updenh

        # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;


        # Menu record number of current menu's predecessor,
        # menu record number of current or saved task menu ,
        # name of active BG Task (if any),
        # and current menu's type
ommon /blk9/ipredm,icurtm,iactsk(TNL), mentyp;



        # Index of permanent button (in inopt)
        # that is displayed in reverse mode
common /blk14/ipenh;


        # array containing input option information
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1), inopt(1,1));

        # if an enhancement exists, check if it has to
        # be reversed back
if(ipenh != 0)
   {
        # menu type of current menu
   kode = inopt(CODE,ipenh)-8;
   if(mentyp == kode) return    # no menu typ change
   else
     call enhanc(ipenh,REVEND);
   }

        # if new menu is task or initial menu, clear ipenh
        # else enhance new button
if(mentyp == TASK | mentyp == START)
   ipenh = 0
else
   {
   kode = mentyp+8;             # type converted into action code
   for(i=1; i<=CONMAX; i=i+1)   # search action code to
                                # determine index in inopt
      {
      j=i;                # save index
      if(inopt(CODE,i) == kode) break;
      }
   call enhanc(j,REV);          # reverse button
```

```
    ipenh = j;                        # save reversed button index
    }
:eturn;
end
```

```
#*****************************************************************
#
#              5.  PARAMETER BLOCK MANAGER
#
#*****************************************************************
#
# Initial release -- hn/16 november 79
# NAME:
#       bgpbm(inx)
# CALLING PROGRAM:
#       bgcmdm
# FUNCTION:
#       update the Parameter Block (PB)
#       save the PB on disc when update is done
#       sends the PB to the Foreground Job
# INPUT PARAMETER:
#       inx - pointer to selected input option in inopt
# INFORMAL INPUT:
#       inopt - info about possible input
# ALGORITHM:
#       get annex from inopt to get new parameter
#       if(done)
#         save PB on disc
#         send PB to Foreground Job
#       else
#         store parameter in PB
#
##################################################################

subroutine bgpbm(inx);

common /blk1/mentbl(TBLMAX), numopt;
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1),inopt(1,1));

#+++++++ modify PB

return;
end
```

```
#*****************************************************************
#
#               6.  LOG MANAGER
#
#*****************************************************************
#
# Initial release -- hn/16 november 79
# NAME:
#       bglogm(index)
# CALLING PROGRAM:
#       bginpm
# FUNCTION:
#       log the operator's input action on a disc file
# INPUT PARAMETER:
#       index -- index to input in array inopt
# INFORMAL INPUT:
#       inopt - contains info about possible inputs
# ALGORITHM:
#       get plain English command from inopt
#       write command to disc file 'inplog.dat' on
#       logical unit LULOG
#
#################################################################

subroutine bglogm(index);

common /blk1/mentbl(TBLMAX), numopt;

common /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1),inopt(1,1));

write(lulog,100) (inopt(i,index),i=2,6);
100 format(5a2);

return;
end
```

```
*********************************************************************
                                                                    *
          7.   M E S S A G E    M A N A G E R
                                                                    *
*********************************************************************

Initial release -- hn/11 Sept 79
NAME:
      bgmsgm(itype)
CALLING PROGRAM:
      EXECUTIVE -- the very first time (requested by RT-11)
      RT-11     -- bgmsgm is to be scheduled using itimer call
FUNCTION:
      Update the enhancement of the ACKNOWLEDGE button
      Process requests to erase/display messages (operator input
         related or warning messages) initiated by the BG Controller
         or any BG Task
      Erase/display messages on the TT Display
      save message stacks on disc/ read msg stacks from disc
# INPUT PARAMETER:
#      itype -- specifies the action requested by the
#               scheduling program
#        NOACTN - dummy call from executive
#        EINPUT - Erase input related message
#        EWARN  - Erase warning message
#        DINPUT - Display input related message
#        DWARN  - Display warning message from Controller
#        DWARNT - Display warning message from Task
# INFORMAL INPUT:
#      msgstk - warning message stack
#      msgstr - read pointer for stack
#      msgstw - write pointer for stack
#      inpmsg - input related message
#      iwmsg  - warning message initiated by a BG Task
#      msgw   - warning message initiated by BG Controller
#      msgwpr - read pointer for msgw
#      msgcnt - counter for warning messages
#  mgstsw - points to msg stack half to write into
# INFORMAL OUTPUT:
#      same as informal input
# NOTE:
#      This is a completion routine and will not be
#      interrupted by any other BG program.
# ALGORITHM:
#      if NOACTN , return
#      if EINPUT
#         erase message field, return
#      if EWARN
#         display any waiting warning message
#         update ACKNOWLEDGE enhancement
#         return
#      if DINPUT
#         display message, return
#      if DWARN or DWARNT
```

```
#         display warning message or save it on stack
#         update ACKNOWLEDGE enhancement
#         return

##########################################################################

ıbroutine bgmsgm(itype);

ommon /blk10/killfl,inflag;

       # no action on dummy call from EXECUTIVE
f(itype == NOACTN) return;

       # accept no input when displaying
nflag=NOACPT;

       # Erase input related message
f(itype == EINPUT)
  {
  call einmsg;
  }

       # Erase warning message
lse if(itype == EWARN)
  {
  call ewmsg;
  }

       # Display input related message
else if(itype == DINPUT)
  {
  call einmsg;
  call dinmsg;
  }

       # Display  warning message
else if(itype == DWARN | itype == DWARNT)
  {
  call dwmsg(itype);
  }
       # accept input
inflag=ACCEPT;

return;
end

#*********************************************************************
#                                                                   *
#              7.1     E I N M S G
#                                                                   *
#*********************************************************************
#
# Initial release -- hn/12 Sept 79
```

```
NAME:
       einmsg
CALLING PROGRAM:
       bgmsgm
FUNCTION:
       Erase any displayed operator input related
       message on the display
ALGORITHM:
       position display cursor
       clear line (ESC K)

##################################################################

ubroutine einmsg;

ommon /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;
       # command string to erase input related message
ogical*1 erasei(12);
       # BEL ESC & a 2 1 r 2 1 C ESC K
lata erasei/07,27,38,97,50,49,114,50,49,67,27,75/;

hp=iwritw(6,erasei,0,luttd);

eturn;
nd




#*****************************************************************
#                                                                *
#              7.2     E W M S G
#                                                                *
#*****************************************************************
#
# Initial release -- hn/12 Sept 79
# NAME:
#      ewmsg
# CALLING PROGRAM:
#      bgmsgm
# FUNCTION:
#      Erase the display of a warning message
#      Display any waiting warning message
#      Update the enhancement of ACKNOWLEDGE button
# INFORMAL INPUT:
#      msgstk - message stack
#      msgstr - read pointer for stack
#      msgstw - write pointer for stack
#      msgcnt - counter for warning messages
#      mgstsw - stack switch, indicating into which stack half
#               a new message has to be stored
#      inopt  - operator input option information
# INFORMAL OUTPUT:
```

```
#       same as informal input
# ALGORITHM:
#       if no message on stack
#          erase message display
#          return
#       while there is message text , do
#          if read pointer in upper half of stack
#             get next characters
#             if ENDOFM, break loop
#          else
#             if no stack saved on disc
#                move lower half to upper half of stack
#             else
#                read stack from disc into upper half
#                update pointer to stacks on disc
#
#       update message counter
#       if no messages on stack
#          stop blinking ACKNOWLEDGE button
#       display message
#       return
#
#######################################################################

subroutine ewmsg

        # Menu Table declaration, number of input options
common /blk1/mentbl(TBLMAX), numopt;


        # Message stack, stack read and write pointer
logical*1 msgstk(MGSTL1);
common /blk8/msgstk, msgstr, msgstw;


        # number of warning messages to be displayed
        # index to warning message stack half to write into
        # first message stack record number on disc file
        # last message stack record number on disc file
common /blk12/msgcnt, mgstsw, mfrec, mlrec;

common /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

        # temporary storage of message text
logical*1 mbuff(MSGL);

        # array to structure the message stack
        # into upper and lower half
logical*1 mstk(MGSTL2,2);
equivalence (msgstk(1), mstk(1,1));

        # array containing input option information
integer*2 inopt(OPTL,INMAX);
equivalence (mentbl(1),inopt(1,1));
```

```
      # command string to erase warning message
)gical*1 erasew(12);
ata erasew/07,27,38,97,50,50,114,50,49,67,27,75/;


      # if only 1 warning message exists, erase it
:(msgcnt <= 1)
  {
  ihp=iwritw(6,erasew,0,luttd);
  msgcnt = 0;
  return;
  }

      # copy a warning message from msg stack into msg buffer
      # if needed msg stack is saved on disc, read it and
      # continue to copy.
buffx=2;  mbuff(1)=36;   # '$'
hile(mbuffx <= MSGL)
  {
      # if msg on stack in memory
  if(msgstr <= MGSTL2)
     {
     mbuff(mbuffx) = msgstk(msgstr);
     msgstr = msgstr+1;
       # stop if message complete
     if(mbuff(mbuffx) == ENDOFM)  break;
     mbuffx=mbuffx+1;
     }

      # warning message either in lower half of message stack
      # or there is a msg-stack saved on disc
  else
     {
      # if no stack on disc
              # copy lower stack half into upper half
     if(mfrec == 0)
        {
        for(i=1; i<=MGSTL2;  i=i+1)
           mstk(i,1) = mstk(i,2);
        mgstsw = 1;
        }
              # if stack on disc, read it into
              # upper stack half
     else
        {
        read(lumsg'mfrec,ERR=9999) (mstk(i,1),i=1,60)
        if(mfrec == mlrec)
           {
           mfrec = 0;
           mlrec = 0;
           }
        else
           mfrec = mfrec+1;
```

```
        }

      # reset read pointer in msg stack
    msgstr = 1;
    }
  }

      # update counter for warning messages
      # if no more messages waiting, stop blinking
      # of ACKNOWLEDGE button
sgcnt = msgcnt-1;
f(msgcnt < 2)
  {
  msgstr = 1;
  msgstw = 1;
  mgstsw = 1;
  for(i=1; i<=CONMAX; i=i+1)
     {
     if(inopt(CODE,i) == ACODAC)
        {
        call enhanc(i,BLKEND);
        break;
        }
     }

  }
      # write message, clear rest of line
hp=iwritw(6,erasew,0,luttd);
buff(mbuffx)=0;           # ASCII NUL
words=mbuffx/2;           # number of words
hp=iwritw(nwords,mbuff,0,luttd);
eturn;

      # if read error
999  continue
eturn;
nd



*********************************************************************
                                                                    *
              7.3    D I N M S G
                                                                    *
*********************************************************************

 Initial release -- hn/26 Sept 79
 NAME:
      dinmsg
 CALLING PROGRAM:
      bgmsgm
 FUNCTION:
      display new input related message
      NOTE:
```

```
#          the first character in a message
#          will not be displayed
#INFORMAL INPUT:
#       inpmsg - input related message
#       inopt  - info about operator input options
#ALGORITHM:
#       position cursor
#       display new message

################################################################################

ıbroutine dinmsg;

        # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;

ogical*1 inpmsg(MSGL), iwmsg(MSGL);
ommon /blk5/inpmsg,iwmsg;

ommon /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

        # array containing input option info
nteger*2 inopt(OPTL,INMAX);
quivalence (mentbl(1), inopt(1,1));
ogical*1 swtext(2);
ata swtext/36,15/;



        # write the message, any old one has already been
        # erased by einmsg, the cursor is positioned
hp=iwritw(1,swtext,0,luttd);
hp=iwritw(27,inpmsg,0,luttd);
eturn;
nd




*******************************************************************************
                                                                              *
              7.4     D W M S G
                                                                              *
*******************************************************************************

 Initial release -- hn/26 Sept 79
 NAME:
       dwmsg(itype)
 CALLING PROGRAM:
       bgmsgm
 FUNCTION:
```

```
#       display a warning message or
#         store it on stack
#       update enhancement of ACKNOWLEDGE button
#INPUT PARAMETER:
#       itype - specifies the message originator
#          DWARN  - msg from BG Controller
#          DWARNT - msg from BG Task
#INFORMAL INPUT:
#       msgstk - message stack
#       msgstr - read pointer for stack
#       msgstw - write pointer for stack
#       msgcnt - counter for warning messages
#       mgstsw - stack switch, indicates into which half of
#                the stack (upper,lower) a new message has to
#                be stored
#       msgw   - warning message from BG Controller
#       msgwpr - read pointer for msgw
#       iwmsg  - warning message from BG Task
#       inopt  - input option information
 INFORMAL OUTPUT:
        same as informal input
 ALGORITHM:
        Store message from either controller or task into buffer
        If no message being displayed
           display new message
           reset stack pointer
   else
           copy mesage from buffer to stack
           if stack overflow
              write stack onto disc
              update pointers and flags
   increment message counter
       if two messages present, start blinking of ACKNOWLEDGE
   return

#####################################################################

ubroutine dwmsg(itype);

       # Menu Table declaration, number of input options
ommon /blk1/mentbl(TBLMAX), numopt;


       # Storage for operator input related messages
       # and warning messages from tasks
ogical*1 inpmsg(MSGL), iwmsg(MSGL);
ommon /blk5/inpmsg, iwmsg;

       # Storage for
       # warning messages from BG Controller,
       # read and write pointer for msgw
ogical*1 msgw(MSGWL);
ommon /blk6/msgw,msgwpr,msgwpw;
```

```
     # Message stack, stack read and write pointer
ical*1 msgstk(MGSTL1);
mon /blk8/msgstk, msgstr, msgstw;


     # number of warning messages to be displayed
     # index to warning message stack half to write into
     # first message stack record number on disc file
     # last message stack record number on disc file
mon /blk12/msgcnt, mgstsw, mfrec, mlrec;

mon /blk17/lumsg,lumstk,lumenu,luttd,lutsys,lulog;

     # temporary storage for messages
gical*1 mbuff(MSGL);

     # array to structure the message stack
     # into upper and lower half
gical*1 mstk(MGSTL2,2);
uivalence (msgstk(1), mstk(1,1));

     # array containing input option information
teger*2 inopt(OPTL,INMAX);
uivalence (mentbl(1), inopt(1,1));

     # array containing command string to position
     # cursor and clear line
gical*1 displw(12);
ta displw/07,27,38,97,50,50,114,50,49,67,27,75/;

     # message generated by BG Controller
(itype == DWARN)
 {
 mbuff(1)=36;  # '$'
 mbuff(2)=15;   # Control O
 for(i=3; i<=MSGL; i=i+1)
    {
    mbuff(i) = msgw(msgwpr);
    msgwpr = msgwpr+1;
    if(msgwpr > MSGWL)  msgwpr=1;
    max = i;           # save index
    if(mbuff(i) == ENDOFM) break;
    }
 }

      # message generated by BG Task
lse
 {
 mbuff(1)=36;  # '$'
 mbuff(2)=15;  # Control O
 for(i=3; i<=MSGL; i=i+1)
    {
     max = i;          # save index
```

```
     mbuff(i) = iwmsg(i);
     if(mbuff(i) == ENDOFM) break;
    }
 }

      # display new message if it is the first one
(msgcnt == 0)
 {
 ihp=iwritw(6,displw,0,luttd);
 nwords=max/2;
 mbuff(max)=0;          # ASCII NUL
 ihp=iwritw(nwords,mbuff,0,luttd);
 msgstr = 1;
 msgstw = 1;
 mgstsw = 1;
 }

      # copy message to stack (upper or lower half)
se
 {
 for(mbuffx=1; mbuffx<=max; mbuffx=mbuffx+1)
    {
              # no overflow of message half
    if(msgstw <= MGSTL2)
       {
       mstk(msgstw,mgstsw) = mbuff(mbuffx);
       msgstw = msgstw+1;
       }
    else
       {
              # if upper switch to lower half of stack
       if(mgstsw == 1)
          mgstsw = 2
       else
              # upper and lower half overflown
              # write lower half onto disc
          {
          if(mfrec == 0)
            {
            mfrec = 1;
            mlrec = 1;
            }
          else
            mlrec = mlrec+1;
      write(lumsg'mlrec,ERR=9999) (mstk(i,2),i=1,60)
          }
       msgstw = 2;
       mstk(1,mgstsw)=mbuff(mbuffx);
       }
    }
 }

      # update message counter and enhancement
gcnt = msgcnt+1;
```

```
(msgcnt == 2)
 {
 for(i=1; i<=CONMAX; i=i+1)
     {
     if(inopt(CODE,i) == ACODAC)
         {
         call enhanc(i,BLKBEG);
         break;
         }
     }
 }

eturn;

       # write error
999 continue
eturn;
nd
```

MENU EDITOR PROGRAM

```
;*********************************************************************
;                                                                   *
;               M E N U E D
;                                                                   *
;*********************************************************************
;
;Initial release -- hn/1 Oct 79
;NAME:
;       menued
 FUNCTION:
        create new menus
        update existing menus
        delete menus
        print menu-id's of all menus stored on the menu file
 CALLING PROGRAM:
        RT-11
 INPUT:
        text file with all needed information having the format
        described below.
 OUTPUT:
        - text file  or
        - new menu on disc  or
        - file with all menu id's
 NOTES:
        This is an off-line program.
        It interprets a standard input in certain format.
        Each @ (end of text), or ',' (end of integer), or
        | (end of line) may be followed by comments.
        Input may be files or any input device. Each line
        of input has to be in the following formats
        ( line number ):

(1)   : function
        2: create - create a new menu
        3: update - update an existing menu
        1: delete - delete an existing file
        8: initm  - create initial start menu

The following formats will depend on the functions.

for all input files:
(1) : function code

for delete, create, update
(2) : menu-id

for create,update
(3)   : predecessor menu-id
(4)   : menu-type
(5)   : menu title
```

```
#(6)  : number of question/text blocks
#
#      for each question/text block:
      : start-block code  (126)
      : number of responses
      : location of buttons
      : size of buttons
      : text

       for each response:
      : start-response code  (127)
      : label (1 characters)
      : plain-English command for logging (10 characters)
      : action code
      : annex

####################################################################


efine  LUR      9        # logical unit for read
efine  LUW      6        # logical unit for write
efine  LUMR     7        # logical unit for menu file read
efine  LUMW     8        # logical unit for menu file write
efine  LUTTD    4        # logical unit  display

efine  NDREC    20       # number of directory records
efine  NIREC    4        # max 4 records for image
efine  NOREC    2        # max 2 records for input options

       # function codes from input file
efine  PREDM    0        # find a menu id in directory
                         # ( id in ipredm)
efine  DELETE   1        # delete menu id in directory
efine  CREATE   2        # insert a new menu id in directory
efine  UPDATE   3        # find a menu id in directory
                         # (id in menuid)
efine  SUCCM    4        # find menuid of successor menu
efine  INITM    8        # create the initial menu


       # code for button locations
efine  LOCA     1        # button row below the text
efine  LOCB     2        # button row beside text
efine  LOCC     3        # button collumn beside text


       # symbolic names for fixed margins
efine  LEFTM    12       # left margin of menu field
efine  LMARG    20       # lower margin of menu field

       # right margins for button rows/collumns
efine  RMARGA   75       # button row below text
efine  RMARGB   75       # button row beside text
efine  RMARGC   75       # button collumn beside text
```

```
fine  DELTA    2        # free rows/coll between buttons
fine  IMAGEL   2432     # image storage length (bytes)
fine  INOPTL   15       # word length for each input option
fine  INMAX    37       # max number of input options
fine  IBUFFL   80       # input buffer length
fine  ENDOFI   8316     # code for end-of-input
fine  ENDOFR   32382    # code for end-of-record
fine  TEXT     15       # code for text mode
fine  BUTT     14       # code for line mode
fine  NOCH     0        # no change of mode
fine  SBLOCK   126      # code for start-text/question-block
fine  SRESP    127      # code for start-response-block

      # symbolic input parameter
fine  DONE     1        # done
fine  NODONE   0        # not yet done

      # ASCII character
fine  ESC      27       # ESC
fine  CONTRO   15       # CONTROL O
fine  CONTRN   14       # CONTROL N


      # input buffer
teger*2 ibuff(IBUFFL);

      # storage for menu-id
teger*2 menuid(10);

      # storage for menu-id to lookup in directory
teger*2 ipredm(10);

      # storage for image
teger*2 image(IMAGEL);

      # storage for input options
teger*2 inopt(INOPTL,INMAX);

      # pointer
teger*2 imagep          # next free entry in image
teger*2 inoptp          # next free entry in inopt

      # next line to write on display
teger*2 lincnt;

      # display mode
teger*2 mode;

      # function code
teger*2 ifunct;

teger*2 idold(10);      # menuid to find in directory
```

```
teger*2 idnew(10);     # menuid to store in directory

      # menu parameter
teger*2 nblock;        # number of text/question blocks
teger*2 nbutt;         # number of buttons for a block
teger*2 iloc;          # location of buttons in a block
teger*2 isize;         # size of buttons
teger*2 mentyp;        # menu type
teger*2 ipredr;        # record number of predecessor menu
teger*2 ispace;        # space lines between text blocks
teger*2 irec,param,ichar;
teger*2 mr(21,50),ptr;         # temporary storage for menuid,rec #
teger*2 rowcur;        # row the cursor is in
teger*2 highr;         # highest record number created
teger*2 ibseq(7);      # sequence of drawing permanent buttons
                       # entries are index to inopt

teger insave,irsave,iwsave;   # file descriptor from open
mmon /blk1/image,imagep,inopt,inoptp;
mmon /blk2/lincnt;
mmon /blk3/mode;
mmon /blk4/menuid,ipredm;
mmon /blk5/ibuff;
mmon /blk6/nblock;
mmon /blk7/nbutt,iloc,isize;
mmon /blk8/mentyp,ipredr;
mmon /blk9/ispace;
mmon /blk10/rowcur;
mmon /blk11/irsave,iwsave;
mmon /blk12/idold,idnew;
mmon /blk13/mr,ptr;
mmon /blk14/highr;
mmon /blk15/ifunct;
mmon /blk16/ibseq;

bseq(1)=1;
bseq(2)=2;
bseq(3)=4;
bseq(4)=7;
bseq(5)=5;
bseq(6)=6;
bseq(7)=3;



      # specify logical unit numbers and
      # open menufiles for read/write
      # the old menu file 'menufile' is read and
      # it is written onto a temporary menu file 'menuout'.
      # after all is done, 'menuout' is copied to 'menufile'.
ll setfil(LUMR,'menufile ');
ll setfil(LUMW,'menuout ');
ll setfil(LUR,'infile ');
save=LUMR;
```

```
save=LUNW;

      # preset inopt,image,menuid,ipredm,ichar,mr
      # imagep,inoptp,lincnt,ispace
r(i=1;i<=INMAX;i=i+1)

or(j=1;j<=INOPTL;j=j+1)
 inopt(j,i)=12336;    # '0'.'0'


r(i=1;i<=IMAGEL;i=i+1)
 image(i)=8240;       # SP.'0'

r(i=1;i<=10;i=i+1)

enuid(i)=8240;        # SP.'0'
predm(i)=8240;        # SP.'0'


har=8256;     # SP.'@'
r=1;
r(i=1;i<=50;i=i+1)
 {
 for(j=1;j<=21;j=j+1)
   mr(j,i)=0;
 }

incnt=0;
noptp=1;
magep=1;
space=0;
owcur=0;


       # read function code
ead(LUR,1000) ifunct;
000 format(i3);

       # create initial menu
f(ifunct == INITM)
      call initm

lse
  {
       # delete,create,update
       # save menu-id
  read(LUR,100) ibuff;
  100 format(80a1);

  i=1;
  while(i<=10 & ibuff(i)!=ichar)
   { menuid(i)=ibuff(i); i=i+1; }

       # delete
```

```
 if(ifunct == DELETE)
      call delete

 else
    {
      # create,update
    if(ifunct == CREATE | ifunct == UPDATE)
      {
      read(LUR,100) ibuff;

      i=1;
      while(i<=10 & ibuff(i)!=ichar)
       { ipredm(i)=ibuff(i); i=i+1;}
      param=PREDM;
      call setmen(param,irec);


      ipredr=irec;
      # read menu type
      read(LUR,1000) mentyp;
      call setmen(ifunct,irec);
      call update(irec);
      }

    }
 }

      # close all files
ewind LUMR;
ewind LUMW;
ewind LUR;

nd
```

```
**********************************************************************

                      S E T M E N

**********************************************************************

Initial release -- hn/2 Oct 79
NAME:
      setmen(iflag,irec)
CALLING PROGRAM:
      menued
FUNCTION:
      prepare store for new and old menuid
      and set menuid into mr
INPUT PARAMETER:
```

```
#      iflag --
#            SUCCM  - find successor menu in directory
#            CREATE - insert a menu id in directory
#            UPDATE - find a menu id in directory
#            PREDM  - find predecessor menu id in directory
#OUTPUT PARAMETER:
#      irec -- index of menuid in mr
#INFORMAL INPUT:
#      menuid - menu id to create or update
#      ipredm - menu id to find in directory
#ALGORITHM:
#      reset temporary id stores
#      copy id's
#      save old and new menuid in mr, return index


###################################################################

ibroutine setmen(iflag,irec);


iteger*2 iflag,irec;
iteger*2 p1,p2;
ommon /blk4/menuid,ipredm;
iteger*2 menuid(10);
nteger*2 ipredm(10);

       # store for old and new menuid
ommon /blk12/idold,idnew;
nteger*2 idold(10);
nteger*2 idnew(10);



       # preset temporary storage for id's
or(i=1; i<=10; i=i+1)
 {
 idold(i)=8240;          # SP.'0'
 idnew(i)=8240;
 }

       # store old and new menuid's
f(iflag==SUCCM | iflag==PREDM)
 {
 for(i=1;i<=10;i=i+1)
   {
   idold(i)=ipredm(i);
   idnew(i)=ipredm(i);
   }
 }
lse
 {
 for(i=1;i<=10;i=i+1)
    {
```

```
  idold(i)=menuid(i);
   idnew(i)=menuid(i);
   }


     # save menuid in mr
.1 savemr(irec);

urn;
}


*******************************************************************

                    C P F I L E

*******************************************************************

Initial release -- hn/6 november 79
NAME:
     cpfile
FUNCTION:
     copy the file menuout (up-to-date) to menufile

##################################################################

broutine cpfile;

teger*2 rbuff(544);

ll cofbck;    # close and open files
     # set exit for END-OF-FILE condition
(ierror(104)!=0) goto 7777;
ile(l>0)      # do until done
{
read(LUMR,100) (rbuff(i),i=1,544);
write(LUMW,100) (rbuff(i),i=1,542);
100 format(544al);
}

77 return;
d


*******************************************************************

                  C O F F W D

*******************************************************************

Initial release -- hn/31 october 79
NAME:
     coffwd
```

```
#FUNCTION:
#       close menufile and menuout
#       open these files
#       read from menufile, write to menuout
#
#ALGORITHM:
#       rewind files
#       connect logical unit number and file
#
###################################################################

sbroutine coffwd;

ommon /blk11/irsave,iwsave;
nteger irsave,iwsave;

ewind LUMR;
ewind LUMW;
save=LUMR; iwsave=LUMW;


all setfil(LUMR,'menufile ');
all setfil(LUMW,'menuout ');


eturn;
nd



**********************************************************************

                    C O F B C K

**********************************************************************

 Initial release -- hn/31 october 79
 NAME:
       cofbck
 FUNCTION:
       close menufile & menuout
       open these files, read from menuout, write to menufile
 ALGORITHM:
       rewind files
       connect logical unit and filenames

###################################################################

ubroutine cofbck;

ommon /blk11/irsave,iwsave;
nteger irsave,iwsave;

ewind LUMR;
```

```
rwind LUMW;
isave=LUMR; iwsave=LUMW;


cll setfil(LUMR,'menuout ');
cll setfil(LUMW,'menufile ');


rturn;
ed




************************************************************************

                    I N I T M

************************************************************************

 Initial release -- hn/12 oct 79
 NAME:
       initm
 FUNCTION:
       generate menu records for the initial menu with permanent
       buttons and store them behind the directory records.
 CALLING PROGRAM:
       menued

########################################################################

ubroutine initm;

ommon /blk1/image,imagep,inopt,inoptp;
ommon /blk4/menuid,ipredm;
ommon /blk14/highr;
ommon /blk16/ibseq;
nteger*2 highr;        # first record number of last menu
nteger*2 ibseq(7);
nteger*2 image(IMAGEL),imagep,inopt(INOPTL,INMAX),inoptp;
nteger*2 menuid(10), ipredm(10);

nteger*2 p1,p2,p3,p4,ir,ic,irec,label(4,7);
nteger*2 l1(6),l2(6),l3(6),l4(6),l5(6),l6(6),l7(6);
nteger*2 rbuff(544);
quivalence (rbuff(1), inopt(1,1));

ata label(1,1)/80/; # P
ata label(2,1)/82/; # R
ata label(3,1)/32/; # SP
ata label(4,1)/49/; # 1

ata label(1,2)/80/; # P
ata label(2,2)/82/; # R
```

```
ta label(3,2)/32/; # SP
ta label(4,2)/50/; # 2

ta label(1,3)/65/; # A
ta label(2,3)/67/; # C
ta label(3,3)/75/; # K
ta label(4,3)/32/; # SP

ta label(1,4)/83/; # S
ta label(2,4)/84/; # T
ta label(3,4)/65/; # A
ta label(4,4)/84/; # T

ta label(1,5)/83/; # S
ta label(2,5)/69/; # E
ta label(3,5)/76/; # L
ta label(4,5)/32/; # SP

ta label(1,6)/72/; # H
ta label(2,6)/69/; # E
ta label(3,6)/76/; # L
ta label(4,6)/80/; # P

ta label(1,7)/67/; # C
ta label(2,7)/84/; # T
ta label(3,7)/82/; # R
ta label(4,7)/76/; # L

ata l1(1)/'p '/;
ata l1(2)/'PR'/;
ata l1(3)/'IN'/;
ata l1(4)/'T '/;
ata l1(5)/'CR'/;
ata l1(6)/'T1'/;

ata l2(1)/'r '/;
ata l2(2)/'PR'/;
ata l2(3)/'IN'/;
ata l2(4)/'T '/;
ata l2(5)/'CR'/;
ata l2(6)/'T2'/;

ata l3(1)/'a '/;
ata l3(2)/'AC'/;
ata l3(3)/'KN'/;
ata l3(4)/'OW'/;
ata l3(5)/'LE'/;
ata l3(6)/'DG'/;

data l4(1)/'s '/;
data l4(2)/'ST'/;
data l4(3)/'AT'/;
data l4(4)/'US'/;
data l4(5)/'  '/;
```

```
d:a 14(6)/'  '/;

d:a 15(1)/'e '/;
dta 15(2)/'SE'/;
dta 15(3)/'LE'/;
dta 15(4)/'CT'/;
dta 15(5)/'  '/;
dta 15(6)/'  '/;

dta 16(1)/'h '/;
dta 16(2)/'HE'/;
dta 16(3)/'LP'/;
dta 16(4)/'  '/;
cta 16(5)/'  '/;
cta 16(6)/'  '/;

cta 17(1)/'c '/;
cta 17(2)/'CO'/;
cta 17(3)/'NT'/;
ita 17(4)/'RO'/;
ita 17(5)/'L '/;
ita 17(6)/'  '/;


      # prepare 'menufile' by writing 21 records
      # of 542 bytes to it (Unix will add CR/LF
      # thus create 544 byte records)
      # These records cover the menu file directory
      # The record will be filled with ASCII '0'

      # write 21 records
or(i=1;i<=21;i=i+1)
 {
 write(LUMR,7000) (rbuff(k),k=1,542);
 7000 format(542a1);
 }

      # close menufile and open again
ewind LUMR;
all setfil(LUMR,'menufile ');


      # set menuid to 'startmenu'
enuid(1)=8307;
enuid(2)=8308;
enuid(3)=8289;
enuid(4)=8306;
enuid(5)=8308;
enuid(6)=8301;
enuid(7)=8293;
enuid(8)=8302;
enuid(9)=8309;
enuid(10)=8224;
```

```
        # place menuid and record number into directory
p1CREATE;
cal setmen(p1,irec);

        # set entries for other initial controller menus
        # into the menu directory
        # STATUS, SELECT, HELP, CONTROL
f(i=1;i<=10;i=i+1)
 nenuid(i)=8224;        # SP

        # 'status0'
mnuid(1)=8307;
mnuid(2)=8308;
mnuid(3)=8289;
nuid(4)=8308;
nuid(5)=8309;
nuid(6)=8307;
nuid(7)=8240;
ll setmen(p1,irec);

        # 'select0'
nuid(1)=8307;
nuid(2)=8293;
nuid(3)=8300;
nuid(4)=8293;
nuid(5)=8291;
nuid(6)=8308;
nuid(7)=8240;
all setmen(p1,irec);

        # 'help0'
enuid(1)=8296;
enuid(2)=8293;
enuid(3)=8300;
enuid(4)=8304;
enuid(5)=8240;
enuid(6)=8224;
enuid(7)=8224;
all setmen(p1,irec);

        # 'control0'
nenuid(1)=8291;
nenuid(2)=8303;
nenuid(3)=8302;
nenuid(4)=8308;
menuid(5)=8306;
menuid(6)=8303;
menuid(7)=8300;
menuid(8)=8240;
call setmen(p1,irec);

call updmr;

call initim;    # generate image to initialize TT Display
```

```
       # set label,log command,button coordinates and
       # action code into inopt
for(i=1;i<=6;i=i+1)    # label
  {
  inopt(i,1)=l1(i);
  inopt(i,2)=l2(i);
  inopt(i,3)=l3(i);
  inopt(i,4)=l4(i);
  inopt(i,5)=l5(i);
  inopt(i,6)=l6(i);
  inopt(i,7)=l7(i);
  }

       #  coordinates of buttons
inopt(8,1)=3;
inopt(10,1)=1;
inopt(8,2)=7;
inopt(10,2)=5;
inopt(8,3)=23;
inopt(10,3)=21;
inopt(8,4)=11;
inopt(10,4)=9;
inopt(8,5)=19;
inopt(10,5)=17;
inopt(8,6)=23;
inopt(10,6)=21;
inopt(8,7)=15;
inopt(10,7)=13;

       # action codes
inopt(11,1)=7;
inopt(11,2)=8;
inopt(11,3)=9;
inopt(11,4)=10;
inopt(11,5)=11;
inopt(11,6)=12;
inopt(11,7)=13;

       # set left and right bounds
for(i=1;i<=7;i=i+1)
  {
  if(i==3)      # acknowledge
    {
    inopt(7,3)=10;
    inopt(9,3)=15;
    }
  else
    {
    inopt(7,i)=1;
    inopt(9,i)=6;
    }
  }
```

```
        # store image for buttons and label
p1=2; p2=TEXT;
for(i=1;i<=7;i=i+1)
  {
  inoptp=ibseq(i);
  call draw;
  ic=inopt(7,inoptp)+1; # collum of label
  ir=inopt(10,inoptp)+1;        # row of label
  call poscur(p1,ir,ic,p2);
  for(k=1;k<=4;k=k+1)   # image
    {
    image(imagep)=label(k,inoptp);
    imagep=imagep+1;
    }
  ic=inopt(9,inoptp);
  ir=inopt(8,inoptp);
  call poscur(p1,ir,ic,p2);   # right lower corner
  image(imagep)=inopt(1,inoptp);
  imagep=imagep+1;
  }

        # convert integer of coordinates and action codes
        # into ASCII coded
for(i=1;i<=7;i=i+1)
  {
  for(k=7;k<=11;k=k+1)
    {
    p1=inopt(k,i);
    call intasc(p1,p2,p3);
    inopt(k,i)=p2*256+p3;
    }
  }

        # roll down screen, row 0 now on top
image(imagep)=27; # ESC
imagep=imagep+1;
image(imagep)=84; # T
imagep=imagep+1;
        # clear all lines of the multi purpose menu field
p2=1; p1=2;  p3=LEFTM; p4=NOCH;
call poscur(p1,p2,p3,p4);
for(i=1;i<=LMARG;i=i+1)
  {
  image(imagep)=27; # ESC
  imagep=imagep+1;
  image(imagep)=75; # K
  imagep=imagep+1;
  image(imagep)=27; # ESC
  imagep=imagep+1;
  image(imagep)=66; # B
  imagep=imagep+1;
  if(i==10)             #separate command string
    {
    image(imagep)=27; # ESC
```

```
    imagep=imagep+1;
    image(imagep)=65; # A
    imagep=imagep+1;
    image(imagep)=36; # $
    imagep=imagep+1;
    }
  }
        # set end of input code
image(imagep)=36;        # $ (CR LF)
imagep=imagep+1;
image(imagep)=ENDOFI;
imagep=imagep+1;

call shift;              # arrange image in buffer
        # store parameters
p1=1; call intasc(p1,p2,p3);
inopt(2,36)=p2*256+p3;   # menu typ (start menu)
p1=7; call intasc(p1,p2,p3);
inopt(3,36)=p2*256+p3;   # number of input options
inopt(15,36)=ENDOFR;     # end of record code

        # write imagerecords
        # temporarily substitute the last entry by '0'
        # to prevent that it is a trailing NUL.
for(ind=542; ind<=IMAGEL; ind=ind+540)
  {
  indbeg=ind-541;
  isave=image(ind); image(ind)=8240;  # SP.'0'
  write(LUMW,1000) (image(i),i=indbeg,ind);
  1000 format(544a1);
  image(ind)=isave;
  }

        # write input option records
write(LUMW,500) (rbuff(i),i=1,271);
write(LUMW,500) (rbuff(i),i=271,541);
500 format(272a2);

        # create records for all predecessor menus
irecn=NDREC+6;  # last record of startmenu
while(irecn < highr)
  {
  for(i=1;i<=6;i=i+1)
    {
    write(LUMW,500) (rbuff(k),k=1,271);  # dummy records
    irecn=irecn+1;
    }
  }


        # copy file from menuout to menufile
call cpfile;

return;
```

```
end





#*********************************************************************
#
#                      I N I T I M
#
#*********************************************************************
#
# Initial release -- hn/6 november 79
# NAME:
#       initim
# FUNCTION:
#       create the image to initialize the TT Display
#
######################################################################

subroutine initim;

common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL),imagep,inopt(INOPTL,INMAX),inoptp;

        # ESC E -- Initialization
#image(imagep)=27;      # ESC
#imagep=imagep+1;
#image(imagep)=69;      # E
#imagep=imagep+1;

        # ESC J -- clear screen
#image(imagep)=27;      # ESC
#imagep=imagep+1;
#image(imagep)=74;      # J
#imagep=imagep+1;

        # ESC ) B -- define character set B
#image(imagep)=27;      # ESC
#imagep=imagep+1;
#image(imagep)=41;      # )
#imagep=imagep+1;
#image(imagep)=66;      # B
#imagep=imagep+1;

        # Image to home the cursor and get
        # rid of "TERMINAL READY"
image(imagep)=27;       # ESC
imagep=imagep+1;
image(imagep)=104;      # h
imagep=imagep+1;
image(imagep)=27; # ESC
imagep=imagep+1;
```

```
image(imagep)=75; # K
imagep=imagep+1;

return;
end


#*******************************************************************
#
#                        D E L E T E
#
#*******************************************************************
#
# NAME:
#       delete
# FUNCTION:
#       delete an entry in menu directory
# ALGORITHM:
#       while(entries in directory)
#         read record into input buffer
#         if entry was already found
#           write buffer
#         else
#           search entry in buffer
#           if found
#             delete entry in buffer
#             set found flag
#             write buffer
#
#       copy rest of menufile to menuout
#       copy menuout to menufile
#
####################################################################

subroutine delete;

common /blk4/menuid,ipredm;
integer*2 menuid(10),ipredm(10);

integer*2 buff(544), dir(12,45), irec,found;
equivalence (dir(1,1), buff(1));

       # preset data
irec=1; # counter for records
found=0;        # found flag -- notfound

       # set END-OF-FILE exit
if(ierror(104) != 0) goto 9999;

       # search directory records
while(irec <= NDREC)
  {
  read(LUMR,1000) (buff(i),i=1,544);
```

```
  1000 format(544a1);
  if(found == 0)                # entry not yet found
    {
    k=1;        # pointer to entries
    while(k<=45 & found==0)
      {
      l=1;      # pointer to characters
      while(l<=10)
        {
        if(menuid(l) != dir(l,k)) break;        # no match
        if(l>=10)                # entry found
          {
          found=1;
          for(i=1;i<=10;i=i+1)
            dir(i,k)=8240;       # fill in '0',delete entry
          }
        l=l+1;
        }
      k=k+1;
      }
    }

  write(LUMW,1000) (buff(i),i=1,542);   # write record
  }

        # copy rest of menufile
while(l>0)
 {
 read(LUMR,1000) (buff(i),i=1,544);
 write(LUMW,1000) (buff(i),i=1,542);
 }

9999 call cpfile;       # copy back from menuout to menufile
return;
end



#*********************************************************************
#
#                        U P D A T E
#
#*********************************************************************
#
# Initial release -- hn/12 oct 79
# NAME:
#       update(irec)
# CALLING PROGRAM:
#       menued
# FUNCTION:
#       Determine length of image needed
#       Generate image and input options and store them
#       in the menu file starting at record irec
# INPUT PARAMETER:
```

```
#       irec - record number in menu file to store first record
# NOTE:
#       This routine is responsible that all menu records
#       are copied to the temporary menu file.
# ALGORITHM:
#       generate header (no image generation)
#       while there are more text/question blocks
#          generate a block (no image generation)
#       close the input file
#       open input file for second run
#       generate header
#       generate blocks having the right spacing
#       write records
#       close files
#
######################################################################

subroutine update(irec);

integer*2 irec;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;
common /blk2/lincnt;
integer*2 lincnt;
common /blk6/nblock;
integer*2 nblock;
common /blk9/ispace;
integer*2 ispace;
integer*2 buff(80), p1,p2,p3,p4,p5;


        # preset data
ispace=0;
lincnt=0;
inoptp=1;
imagep=1;


        # generate header,read number of blocks
        # first run - no image
p1=1;
call header(p1);

        # generate blocks, first run - no image -
for(i=0; i<=nblock; i=i+1)
   call block(p1);

        # close input file, open again
rewind LUR;
call setfil(LUR,'infile ');
```

```
        # preset data for second run
if(lincnt>20) write(6,19);19 format('Too many image lines needed');
ispace=(LMARG-lincnt)/(nblock+1);
lincnt = 0;
imagep=1;
inoptp=1;
p1=2;   # second run -- generate image

        # clear all lines of the multi purpose menu field
p2=1; p3=LEFTM; p4=NOCH;
call poscur(p1,p2,p3,p4);
for(i=1;i<=LMARG;i=i+1)
  {
  image(imagep)=27; # ESC
  imagep=imagep+1;
  image(imagep)=75; # K
  imagep=imagep+1;
  image(imagep)=27; # ESC
  imagep=imagep+1;
  image(imagep)=66; # B
  imagep=imagep+1;
  if(i==10)             #separate command string
    {
    image(imagep)=27; # ESC
    imagep=imagep+1;
    image(imagep)=65; # A
    imagep=imagep+1;
    image(imagep)=36; # $
    imagep=imagep+1;
    }
  }


        # update directory records,retrieve data,store it in mr
call updmr;
        # dummy read of function code,menu id,
        # predecessor menu id, menu type
read(LUR,1000) buff(1);
read(LUR,2000) buff;
read(LUR,2000) buff;
read(LUR,1000) buff(1);
1000 format(i3); 2000 format(80a1);

        # generate header -image-
call header(p1);

        # generate blocks -image-
for(i=1;i<=nblock;i=i+1)
   call block(p1);

        # write records, close menu file
call finish(irec);
return;
end
```

```
#***********************************************************************
#
#                          H E A D E R
#
#***********************************************************************
#
# Initial release --hn/12 oct 79
# NAME:
#       header(inp)
# CALLING PROGRAM:
#       update
# FUNCTION:
#       read menu title and number of blocks
# INPUT PARAMETER:
#       inp - 1: first run, no image generation
#             2: second and final run
# ALGORITHM:
#       clear all tabs
#       position cursor to first line
#       copy text
#       read number of blocks
#
########################################################################

subroutine header(inp);


common /blk6/nblock;
integer*2 nblock;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL),imagep,inopt(INOPTL,INMAX),inoptp;

integer*2 inp;
integer*2 p1,p2,p3,dummy;


        # clear all tabs -- ESC 3
image(imagep)=27;       # ESC
imagep=imagep+1;
image(imagep)=51;       # 3
imagep=imagep+1;

        # position cursor for menu title and copy it
p1=0; p2=LEFTM; p3=TEXT;
call poscur(inp,p1,p2,p3);
call copy(inp,dummy);

        # read number of blocks from input file
read(LUR,1000) nblock;
```

```
1000 format(i3);

return;
end






#***********************************************************************
#
#                          B L O C K
#
#***********************************************************************
#
# Initial release -- hn/12 oct 79
# NAME:
#       block(inp)
# CALLING PROGRAM:
#       update
# FUNCTION:
#       read the parameter for a block and store them
#       determine positions of buttons
#       draw each button and label it
# INPUT PARAMETER:
#       inp - 1: first run -no image-
#             2: second and final run
# ALGORITHM:
#       read block parameters
#       copy text/question
#       determine coordinates of buttons
#       draw each button
#
########################################################################

subroutine block(inp);


common /blk2/lincnt;
integer*2 lincnt;
common /blk7/nbutt,iloc,isize;
integer*2 nbutt,iloc,isize;
common /blk9/ispace;
integer*2 ispace;

integer*2 inp;
integer*2 icode;
integer*2 p1,p2,p3,is,ix,iy,ic,lbutt,ltext;



        # read block-start-code
        # if not present write error message
```

```
read(LUR,1000) icode;
1000 format(i3);
if(icode != SBLOCK)
   {
   write(LUW,100);

   100 format(1x,'No text or question block');
   return;
   }

        # read block parameters
else
   {
   read(LUR,1000) nbutt;        # number of buttons
   read(LUR,1000) iloc;         # location of buttons
   read(LUR,1000) isize;        # size of buttons

        # do spacing before block
   is=ispace;
   call linefd(inp,is);
        # button row beside text
   if(iloc == LOCB)
      {
      lbutt=isize+2;
      ltext=lbutt/2;
      ix=RMARGB;
      iy=lincnt+lbutt-1-isize;
      idx=(isize+DELTA)*1.5;
      ix=ix-(nbutt-1)*idx;
      idy=0;
      }

        # button collumn beside text
   else if(iloc == LOCC)
      {
      lbutt=nbutt*(isize+DELTA)-DELTA+1;
      ltext=lbutt/2;
      ix=RMARGC;
      iy=lincnt+lbutt-1-isize;
      idx=0;
      idy=(isize+DELTA);
      iy=iy-(nbutt-1)*idy;
      }

        # button row under text
   else if(iloc == LOCA)
      {
      lbutt=isize+3;
      ltext=1;
      ix=(RMARGA+LEFTM-DELTA+nbutt*(isize+DELTA))/2;
      iy=lincnt+3;
      idx=(isize+DELTA)*1.5;
      ix=ix-(nbutt-1)*idx;
      idy=0;
```

```
     }

        # copy text
   ic = NODONE;
   call linefd(inp,ltext); # linefeed to start of text
   while(ic == NODONE)
      {
      pl=LEFTM; p2=TEXT; p3=lincnt;
      call poscur(inp,p3,pl,p2);
      call copy(inp,ic);
      pl=lbutt-ltext;
      if(ic==NODONE & iloc==LOCA)
        {
        lbutt=lbutt+l;
        iy=iy+l;
        }
      else
         call linefd(inp,pl);   # linefeed below text
      }

        # draw buttons
   for(k=l;k<=nbutt;k=k+l)
      {
      pl=isize;
      call button(inp,pl,ix,iy);
      ix=ix+idx;
      iy=iy+idy;
      }
   }

return;
end




#**********************************************************************
#
#                        F I N I S H
#
#**********************************************************************
#
# Initial release -- hn/12 oct 79
# NAME:
#       finish(irec)
# CALLING PROGRAM:
#       update
# FUNCTION:

#       write menu records onto menu file
# INPUT PARAMETER:
#       irec - first menu record on file
# ALGORITHM:
#       divide image store into record size pieces
```

```
#        if image pointer within a piece, store ENDOFI
#        if image pointer within or at the end of a piece
#           store ENDOFR
#        write record
#        determine last entry in input option store
#        store menu parameter and ENDOFR
#        write record
#
#################################################################################

subroutine finish(irec);


integer*2 imager, inputr,p1,p2,p3,p4,irec;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;
common /blk2/lincnt;
integer*2 lincnt;
common /blk8/mentyp,ipredr;
integer*2 mentyp,ipredr;
common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;
common /blk14/highr;
integer*2 highr;

integer*2 reccnt;        # counter for written records
integer*2 save;
integer*2 inputo(542);
equivalence(inputo(1), inopt(1,1));

        # temporary buffer for records
integer*2 rbuff(544);

reccnt=NDREC;   # preset counter (20 directory records)
        # save first record numbers for image and input options
imager=mr(21,irec);
inputr=imager+NIREC;


        # set END-OF-INPUT code
image(imagep)=ENDOFI;
imagep=imagep+1;

        # shift image to avoid records cutting command strings
        # and set ENDOFR in last image record
call shift;

        # set label for END-OF-FILE condition
if(ierror(104) != 0) goto 9999;
        # copy all records up to first record of this menu
for(k=NDREC+1; k<imager; k=k+1)
```

```
  {
  read(LUMR,1000) (rbuff(ij),ij=1,544);
  write(LUMW,1000) (rbuff(ij),ij=1,542);
   reccnt=reccnt+1;
  1000 format(544a1);
  }

        # write image records
        # temporarily substitute the last character to ensure
        # that it is no space.This would decrease the
        # fixed record length.
for(ind=542;ind<=IMAGEL;ind=ind+540)
  {
  save=image(ind); image(ind)=8240;     # SP '0'
  indbeg=ind-541;
  write(LUMW,1000) (image(i),i=indbeg,ind);
  image(ind)=save;
   reccnt=reccnt+1;
  }



pl=mr(21,ipredr); call intasc(pl,p2,p3);
inopt(1,29)=p2*256+p3;
pl=mentyp; call intasc(pl,p2,p3);
inopt(2,29)=p2*256+p3;
pl=inoptp-1; call intasc(pl,p2,p3);
inopt(3,29)=p2*256+p3;  # number of options (incl. perm.b.)
inopt(15,29)=ENDOFR;


        # convert integer into ASCII
for(k=1;k<inoptp; k=k+1)
 {
 for(kk=7;kk<=11; kk=kk+1)
  {
  pl=inopt(kk,k); call intasc(pl,p2,p3);
  inopt(kk,k)=p2*256+p3;
  }
 }

        # write input option records
write(LUMW,500) (inputo(i), i=1,271);
reccnt=reccnt+1;
write(LUMW,500) (inputo(i), i=271,541);
reccnt=reccnt+1;
500 format(271a2);

        # copy rest of menu file
        # if done goto 9999
if(ierror(104) != 0) goto 9999;
        # dummy read over old menu records
for(i=NIREC+NOREC; i>0; i=i-1)
  read(LUMR,1000) (rbuff(ij),ij=1,544);
```

```
iconst=1;
while(iconst==1)
  {
  read(LUMR,1000) (rbuff(ij),ij=1,544);
  write(LUMW,1000) (rbuff(ij),ij=1,542);
   reccnt=reccnt+1;
  }

9999 while(highr > reccnt)      # more records to create
       {
       for(i=1;i<=6;i=i+1)
          write(LUMW,1000) (rbuff(ij),ij=1,542);
       reccnt=reccnt+6;
       }
call cpfile;    # copy file from menuout to menufile

return;
end




#*********************************************************************
#
#                    C O P Y
#
#*********************************************************************
#
# Initial release -- hn/14oct 79
# NAME:
#       copy(inp,iflag)
# FUNCTION:
#       copy text from input file to image array
# INPUT PARAMETER:
#       inp   - 1: do not copy
#               2: do copy
# OUTPUT PARAMETER:
#       iflag - DONE:  '@' encountered, end of text
#               NODONE: not yet end of text
# ALGORITHM:
#       if inp = 1: return
#       preset iflag to NODONE
#       read 80 character input line
#       if '@'
#          set iflag to DONE, return
#       if 'CR'
#          return
#       copy input character by character
#
######################################################################

subroutine copy(inp,iflag);
```

```
integer*2 inp,iflag,ichar,pl;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;
common /blk5/ibuff;
integer*2 ibuff(80);


        # preset output flag
iflag=NODONE;

        # read input line
read(LUR,100) ibuff;
100 format(80a1);

        # copy characters, break if '@'
ichar=8256;     # SP.'@'
pl=1;
for(i=1; i<=80; i=i+1)
   {
   if(ibuff(i)==ichar)  # SP.'@'
      {
      iflag=DONE;
       call linefd(inp,pl);
      return;
      }
    if(ibuff(i)==8205 | ibuff(i)==8316)  # SP.CR or | (end of line)
      {
       call linefd(inp,pl);
       return;
       }
    else if(inp != 1) {
   image(imagep)=ibuff(i);
   imagep=imagep+1; }
   }

return;
end




#************************************************************************
#
#                L I N E F D
#
#************************************************************************
#
# Initial release -- hn/14 oct 79
# NAME:
```

```
#        linefd(inp,num)
# FUNCTION:
#        generate display commands for line feed/ carriage return
# INPUT PARAMETER:
#        inp - 1: do not generate, only update lincnt
#              2: generate line feeds, update lincnt
#        num - number of line feeds
# ALGORITHM:
#        update lincnt
#        if inp = 1  return
#        generate linefeeds, carriage return
#
##########################################################################

subroutine linefd(inp,num);


integer*2 inp,num;
common /blk2/lincnt;
integer*2 lincnt;

lincnt=lincnt+num;
if(inp==1) return;  # no image generation
call crlf;
return;
end




#***********************************************************************
#
#                P O S C U R
#
#***********************************************************************
#
# Initial release -- hn/14 oct 79
# NAME:
#        poscur(inp,irow,icol,icode)
# FUNCTION:
#        position the cursor and switch to requested mode
# INPUT PARAMETERS:
#        inp   - 1: do nothing
#                2: do
#        irow  - row number
#                -1 if only collumn change
#        icol  - collumn number
#                -1 if only row change
#                if both irow and icol = -1, only change mode
#        icode - new mode
#                BUTT - Alternate character set (lines)
#                TEXT - Base character set
#                NOCH - No change, leave old character set
# ALGORITHM:
```

```
#       if inp=1  return
#       if new mode != NOCH
#            switch to new mode
#       position cursor
#
#######################################################################

subroutine poscur(inp,irow,icol,icode);


integer*2 inp,irow,icol,icode;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;
common /blk3/mode;
integer*2 mode;
common /blk10/rowcur;
integer*2 rowcur;

integer*2 irowmv(6);
integer*2 icolmv(6);
integer*2 ircmv(9);

        # ESC & a _ _ R
data irowmv(1)/27/;
data irowmv(2)/38/;
data irowmv(3)/97/;
data irowmv(4)/32/;
data irowmv(5)/32/;
data irowmv(6)/82/;
        # ESC & a _ _ r _ _ C
data ircmv(1)/27/;
data ircmv(2)/38/;
data ircmv(3)/97/;
data ircmv(4)/32/;
data ircmv(5)/32/;
data ircmv(6)/114/;
data ircmv(7)/32/;
data ircmv(8)/32/;
data ircmv(9)/67/;
        # ESC & a _ _ C
data icolmv(1)/27/;
data icolmv(2)/38/;
data icolmv(3)/97/;
data icolmv(4)/32/;
data icolmv(5)/32/;
data icolmv(6)/67/;

if(inp == 1) return;


        # move only row
```

```
if(icol == -1 & irow != -1)
   {
   irowmv(4) = irow/10+48;        # '0'
   irowmv(5) = mod(irow,10) + 48;         # '0'
   for(i=1; i<=6; i=i+1)
      {
      image(imagep) = irowmv(i);
      imagep=imagep+1;
      }
        # save row of cursor
   rowcur=irow;
   }

        # move only col
else if(irow == -1 & icol != -1)
   {
   icolmv(4) = icol/10 + 48;      # '0'
   icolmv(5) = mod(icol,10) + 48;         # '0'
   for(i=1; i<=6; i=i+1)
      {
      image(imagep) = icolmv(i);
      imagep=imagep+1;
      }
   }

        # move row and col
else if(icol != -1 & irow != -1)
   {
        # separate command string
   if(rowcur<23) call crlf;
   ircmv(4) = irow/10 + 48;       # '0'
   ircmv(5) = mod(irow,10) + 48;          # '0'
   ircmv(7) = icol/10 + 48;       # '0'
   ircmv(8) = mod(icol,10) + 48;          # '0'
   for(i=1; i<=9; i=i+1)
      {
      image(imagep)=ircmv(i);
      imagep=imagep+1;
      }
        # save row of cursor
   rowcur=irow;
   }

        # change mode if necessary
if(icode != NOCH)
   {
   image(imagep) = icode;
   imagep=imagep+1;
   mode = icode;
   }

return;
end
```

```
#**********************************************************************
#
#                B U T T O N
#
#**********************************************************************
#
# Initial release -- hn/ 14 oct 79
# NAME:
#       button(inp,isize,ix,iy)
# FUNCTION:
#       read additional button information and store it in inopt
#       initiate drawing of the buttons
# INPUT PARAMETERS:
#       inp   - 1:do not draw buttons
#               2: do everything
#       isize - size of the buttons to draw (number of lines)
#       ix    - collum number of rigth upper corner
#       iy    - row number of rigth upper corner
# ALGORITHM:
#       read start-of-response code
#       if not there, write error message
#       read and store:
#          label
#          plain-English mesage for logging
#          action code
#       if display-new-menu
#          read menu-id
#          get first record number of menu
#          store record number in inopt
#       if return-character
#          read character and store it
#       if return-integer or return-task-id
#          read integer and store it
#       if parameter-block-update
#          #+++++++ do not know yet
#       if inp == 2
#          store button coordinates
#          create button images
#       increment pointer in inopt
#
######################################################################

subroutine button(inp,size,ix,iy);

integer*2 inp,size,ix,iy,ichar,irec;
integer*2 p1,p2,p3;

common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
```

```
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;

common /blk4/menuid,ipredm;
integer*2 menuid(10);
integer*2 ipredm(10);

common /blk5/ibuff;
integer*2 ibuff(IBUFFL);

integer*2 icode;         # temporary storage

        # input buffer 40 words
integer*2 iwbuff(40);

ichar=8256;     # SP.'@'
        # read start-response code
read(LUR,1000) icode;
 1000 format(i3);
if(icode != SRESP)
   {
   write(LUW,100);
   100 format(1x,'No response to read');
   return;
   }

else
   {
        # read label and store it
   read(LUR,200) ibuff;
   300 format(40a2);
   inopt(1,inoptp) = ibuff(1);

        # read plain-English message and store it
   read(LUR,300) iwbuff;
   for(i=2; i<=6; i=i+1)
       { if(iwbuff(i-1)==ichar) break;
      inopt(i,inoptp) = iwbuff(i-1); }

        # read and store action code
   read(LUR,1000) iact;
   inopt(11,inoptp) = iact;

        # display-new-menu
   if(iact == 1)
      {
        # read menu-id and store it
      read(LUR,200) ibuff;
      200 format(80a1);
      for(i=1; i<=10; i=i+1)
         ipredm(i)=8240;        # SP.'0'
       i=1;
       while(i<=10 & ibuff(i)!= ichar)
```

```
        { ipredm(i)=ibuff(i); i=i+1; }
        if(inp==1)
       { p1=PREDL;
      call setmen(p1,irec);}    # set menu record number
       else { call getmen(irec);

       p1=irec; call intasc(p1,p2,p3);
      inopt(12,inoptp) = p2*256+p3;      # store record number
        }
      }

        # return-character
   else if(iact == 2 | iact == 4)
      {
      read(LUR,300) iwbuff;
      inopt(12,inoptp) = iwbuff(1);
      }

        # integer or task-id
   else if(iact == 3 | iact == 14)
      {
      read(LUR,1000) p1
        call intasc(p1,p2,p3);
         inopt(12,inoptp)=p2*256+p3;
      }

        # ++++++ pb change, do not yet know
   else if(iact == 15)
      {
   continue;#++++ do not know
     }

        # draw buttons
   if(inp == 2)
      {
      inopt(7,inoptp) = ix+1-size*1.5;
      inopt(8,inoptp) = iy-1+size;
      inopt(9,inoptp) = ix;
      inopt(10,inoptp)= iy;

      call draw;
      }

   inoptp=inoptp+1;
   }

return;
end


#**********************************************************************
```

```
#
#                D R A W
#
#***********************************************************************
#
# Initial release --hn/ 15 Oct 79
# NAME:
#       draw
# FUNCTION:
#       create image to draw a button
# ALGORITHM:
#       clear all tabs
#       set tab at right button margin
#       set tab at left button margin
#       position cursor to left upper corner
#       while more lines
#          enhance line within tabs
#          tab to beginning of next line
#          tab to left limit of button
#       draw label
#
#######################################################################

subroutine draw;


integer*2 p1,p2,p3,p4,iy,ir,ic;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;
common /blk10/rowcur;
integer*2 rowcur;

integer*2 icltab(2);    # clear all tabs
integer*2 endenh(4);    # end of enhancement

        # ESC 3
data icltab(1)/27/;
data icltab(2)/51/;

        # ESC & d @
data endenh(1)/27/;
data endenh(2)/38/;
data endenh(3)/100/;
data endenh(4)/64/;

        # separate command string
call crlf;

        # clear all tabs
for(i=1; i<=2; i=i+1)
   {
```

```
   image(imagep)=icltab(i);
   imagep=imagep+1;
   }


        # set right tab
p1=inopt(9,inoptp)+1;
call settab(p1);

        # same for left margin
p1=inopt(7,inoptp);
call settab(p1);

        # save upmost line of button
iy=inopt(10,inoptp);

        # draw enhancement for button
while(iy <= inopt(8,inoptp))
   {
        # position cursor to upper left corner, change mode
   p4=2;  p3=BUTT;
   call poscur(p4,iy,p1,p3);

   call enhanc;
   call tab;
   for(i=1;i<=4;i=i+1)  # end of enhancement command
    {
    image(imagep)=endenh(i);
    imagep=imagep+1;
    }
    iy=iy+1;
   }
        # draw label
ic = (inopt(7,inoptp)+inopt(9,inoptp))/2;
ir = (inopt(8,inoptp)+inopt(10,inoptp))/2;

if(ic<=12) return;      # no label for permanent buttons
p1=2; p2=TEXT;
call poscur(p1,ir,ic,p2);

        # copy label
image(imagep)=inopt(1,inoptp);
imagep=imagep+1;
image(imagep)=inopt(1,inoptp)/256;
imagep=imagep+1;

return;
end



#*****************************************************************
#
```

```
#                S E T T A B
#
#*****************************************************************
#
# Initial release --hn/ 15 oct 79
# NAME:
#       settab(icol)
# FUNCTION:
#       position cursor to specified collumn
#       set tab
# ALGORITHM:
#       move cursor to collumn
#       set tab
#
##################################################################

subroutine settab(icol);


integer*2 icol,p1,p2,p3;
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;

p1=2; p2=-1; p3=NOCH;
call poscur(p1,p2,icol,p3);
image(imagep)=ESC;
imagep=imagep+1;
image(imagep)=49;       # '1'
imagep=imagep+1;
return;
end




#*****************************************************************
#
#                T A B
#
#*****************************************************************
#
# Initial release -- hn/ 15 oct 79
# NAME:
#       tab
# FUNCTION:
#       generate tab
#
##################################################################

subroutine tab;
```

```
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;

image(imagep)=ESC;
imagep=imagep+1;
image(imagep)=73;        # 'I'
imagep=imagep+1;
return;
end




#***********************************************************************
#
#                E N H A N C
#
#**********************************************************************
#
# Initial release --hn/ 15 oct 79
# NAME:
#       enhanc
# FUNCTION:
#       start enhancement for button (inverse video,half bright)
#
########################################################################

subroutine enhanc;


common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL);
integer*2 imagep;
integer*2 inopt(INOPTL,INMAX);
integer*2 inoptp;

integer*2 ienh(4);
        # ESC & d J
data ienh(1)/27/;
data ienh(2)/38/;
data ienh(3)/100/;
data ienh(4)/74/;

for(i=1; i<=4; i=i+1)
   {
   image(imagep)=ienh(i);
   imagep=imagep+1;
   }
```

```
return;
end



#**********************************************************************
#
#               I N T A S C
#
#**********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#       intasc(inp,out1,out2)
# FUNCTION:
#       Conversion of an integer (0-900) into two integer*2 ASCII codes
# ALGORITHM:
#       out1= inp/30 +40 (range ASCII 40-70)
#       out2= mod(inp,30) +40 (range ASCII 40-69)
#
######################################################################

subroutine intasc(inp,out1,out2);

integer*2 inp,out1,out2;
integer num1,num2;

num1=inp;
out1=inp/30+40;
num2=mod(num1,30);
out2=num2+40;
return;
end



#**********************************************************************
#
#               A S C I N T
#
#**********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#       ascint(in1,in2,outp)
# FUNCTION:
#       Conversion of an ASCII coded number (0-900) into
#       an integer*2 value
# ALGORITHM:
#       outp= (in1-40)*30 + (in2-30)
#
######################################################################
```

```
subroutine ascint(in1,in2,outp);

integer*2 in1,in2,outp;

outp=(in1-40)*30+in2-40;
return;
end




#*********************************************************************
#
#                    C R L F
#
#*********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#        crlf
# FUNCTION:
#        create cr/lf for image array without incrementing
#        the linecounter lincnt
#
######################################################################

subroutine crlf;

common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL), imagep;
common /blk10/rowcur;
integer*2 rowcur;


if(rowcur>22) return;  # cr/lf would roll up the screen
  image(imagep)=36;      # $ (CR LF)
  imagep=imagep+1;
return;
end




#*********************************************************************
#
#                    S H I F T
#
#*********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#        shift
```

```
# FUNCTION:
#       shift the contence of the image array to prevent
#       separation of a command string by any record
#       place ENDOFR into last used record
# ALGORITHM:
#       while(image fills another record)
#         start at end of record and search CR/LF
#         if found compute difference to end of record
#         shift rest of image by the difference
#       place ENDOFR into last used record
#
######################################################################

subroutine shift;

common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL),imagep;
integer*2 ind,diff;

        # check all record items
for(ind=540;ind<imagep;ind=ind+540)
  {
  for(index=ind;index>0; index=index-1)
    {
    if(image(index)==ENDOFI) break;  #end of input
    if(image(index)==36)                # $ (CR LF)
      {
      diff=ind-index;
      imagep=imagep+diff;
      for(i=imagep;i>ind;i=i-1)
         image(i)=image(i-diff);
      image(index+1)=ENDOFI;
      break;
      }
    }
  }

        # store ENDOFR
image(ind)=ENDOFR;
return;
end


#*********************************************************************
#
#                        S A V E M R
#
#*********************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       savemr(index)
# FUNCTION:
#       store old and new menuids (idold,idnew) into mr and
```

```
#       return the index in mr
# OUTPUT PARAMETER:
#       index - index in mr
#
##########################################################################

subroutine savemr(index);

integer*2 index;
common /blk12/idold,idnew;
integer*2 idold(10), idnew(10);

common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;

for(i=1;i<=10;i=i+1)
  {
  mr(i,ptr)=idold(i);
  mr(i+10,ptr)=idnew(i);
  }
index=ptr;
ptr=ptr+1;
return;
end


#************************************************************************
#
#                         G E T M E N
#
#************************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       getmen(irec)
# FUNCTION:
#       get the menu record number for a menu
#       whose menuid is in ipredm
# OUTPUT PARAMETER:
#       irec -- record number of first menu record
#
#########################################################################

subroutine getmen(irec);
integer*2 irec;

common /blk4/menuid, ipredm;
integer*2 menuid(10), ipredm(10);

common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;

irec=0; # preset
i=1;    # index in mr
```

```
while(i<ptr & irec==0)
  {
  j=11; # index for first character of menuid
  while(irec==0 & j<=20)
    {
    if(mr(j,i) != ipredm(j-10)) break
    else if(j>=20)
      { irec=mr(21,i); break;}
    else j=j+1;
    }
  i=i+1;
  }

return;
end




#**********************************************************************
#
#                         U P D M R
#
#**********************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       updmr
# FUNCTION:
#       read all menu directory records from menufile and write
#       them to menuout
#       retrieve record numbers for menuid's stord in mr
#       and store them in mr.
# ALGORITHM:
#       open files for read from menufile, write to menuout
#       while there are more directory records
#         read a record
#         for all entries in record
#         compare it with mr entry
#         if found store record number
#         update directory record (new menuid & rec number)
#       if still entries in mr
#         create new entries in directory
#         write record to menuout
#
######################################################################

subroutine updmr;

common /blk13/mr,ptr;
integer*2 mr(21,50), ptr;
common /blk14/highr;
integer*2 highr;
```

```
common /blk15/ifunct;
integer*2 ifunct;

integer*2 rbuff(544), dir(12,45), p1,p2,p3;
equivalence (rbuff(1),dir(1,1));

write(6,13);13 format('updmr: ');
        # close and open files
call coffwd;
if(ierror(104) != 0) goto 9999; # if END-OF-FILE
icount=ptr-1;    # number of entries
highr=0;         # preset

for(i=1;i<=NDREC;i=i+1) # for all directory records
  {
  read(LUMR,100) (rbuff(k),k=1,544);
  100 format(544a1);

  if(icount!=0) {
  for(j=1;j<=45;j=j+1)  # for all record entries
    {
     if(dir(1,j)==8240) # first empty entry
              {
              for(k=1;k<ptr;k=k+1)
                {
                if(mr(21,k)==0)
                  {
                   if((k==2&(ifunct==CREATE|ifunct==INITM)) |
                  (k!=2 & dir(11,j)==8240 & dir(12,j)==8240))
                  { for(kk=1;kk<=10;kk=kk+1)
                       mr(kk,k)=8240;  # old id's now empty
                         break; # only blank the next one
                   }
                     }
                 }
               }

    for(n=1;n<ptr;n=n+1)        # for all mr entries
      {
      for(l=1;l<=10;l=l+1)      # for all characters of menu-id
       {
       if(mr(21,n)!=0) break;  # if record number found
       if(mr(1,n)!= dir(1,j)) break  #char dont match
       else if(l==10)     # menuid found
         {
          for(ll=1;ll<=10;ll=ll+1)  # copy new menuid
            dir(ll,j)=mr(ll+10,n);
          p3=NDREC+1+((i-1)*45+j-1)*(NIREC+NOREC);  # rec number
           if(p3>highr) highr=p3;
          mr(21,n)=p3;
          call intasc(p3,p1,p2);        # convert into ASCII coded
          dir(11,j)=p1; dir(12,j)=p2;
           icount=icount-1;
          }
```

```
          }
        }
      }
     }
   write(LUMW,200) (rbuff(k),k=1,542);
   200 format(542a1);
   }

9999 if(icount>0)
    {
    write(LUW,777); stop;   # not all menu names found
                            # error condition
    777 format(' Not all menus found ,check input -- stop');
    }
return;
end
```

MENU CHECKER PROGRAM

```
#**********************************************************************
#                                                                     *
#              1.  M C H E C K
#                                                                     *
#**********************************************************************
#
# Initial release -- hn/1 Oct 79
# NAME:
#       mcheck
# FUNCTION:
#       print image of a menu
#       print input options of a menu
#       print directory records
# CALLING PROGRAM:
#       RT-11
# INPUT:
#       text file with all needed information having the format
#       described below.
# NOTES:
#       This is an off-line program.
#       It interprets a standard input in certain format.
#       Input may be files or any input device. Each line
#       of input has to be in the following formats
#       ( line number ):
#
# (1)  : function
#       6 = primag - print image of a menu
#       7 = propt  - print input options of a menu
#       5 = prdir  - print directory records
#
# The following formats will depend on the functions.
#
# for all input files:
# (1) : function code
#
# for primag,propt
# (2) : menu-id (10 character)
#
######################################################################


define  LUR     5       # logical unit for read
define  LUW     6       # logical unit for write
define  LUMR    7       # logical unit for menu file read
define  LUMW    8       # logical unit for menu file write
define  LUTTD   4       # logical unit  display

define  NDREC   20      # number of directory records
define  MIREC   4       # max 4 records for image
define  MOREC   2       # max 2 records for input options
```

```
        # function codes from input file
define  PREDM    0        # find a menu id in directory
                          # ( id in ipredm)
define  DELETE   1        # delete menu id in directory
define  CREATE   2        # insert a new menu id in directory
define  UPDATE   3        # find a menu id in directory
                          # (id in menuid)
define  SUCCM    4        # find menuid of successor menu
define  PRINTD   5        # print the contents of directory
define  PRINTI   6        # print image of a menu
define  PRINTO   7        # print input options of a menu
define  INITM    8        # create the initial menu


        # code for button locations
define  LOCA     1        # button row below the text
define  LOCB     2        # button row beside text
define  LOCC     3        # button collumn beside text


        # symbolic names for fixed margins
define  LEFTM    8        # left margin of menu field
define  LEFTP    2        # left margin of permanent buttons
define  LMARG    20       # lower margin of menu field

        # right margins for button rows/collumns
define  RMARGA   78       # button row below text
define  RMARGB   78       # button row beside text
define  RMARGC   78       # button collumn beside text


define  DELTA    2        # free rows/coll between buttons
define  IMAGEL   2432     # image storage length (bytes)
define  INOPTL   15       # word length for each input option
define  INMAX    37       # max number of input options
define  IBUFFL   80       # input buffer length
define  ENDOFI   8316     # code for end-of-input
define  ENDOFR   8318     # code for end-of-record
define  TEXT     15       # code for text mode
define  BUTT     14       # code for line mode
define  SBLOCK   126      # code for start-text/question-block
define  SRESP    127      # code for start-response-block

        # symbolic input parameter
define  DONE     1        # done
define  NODONE   0        # not yet done

        # ASCII character
define  ESC      27       # ESC
define  CONTRO   15       # CONTROL O
define  CONTRN   14       # CONTROL N
```

```
        # input buffer
integer*2 ibuff(IBUFFL);

        # storage for menu-id
integer*2 menuid(10);

        # storage for menu-id to lookup in directory
integer*2 ipredm(10);

        # storage for image
integer*2 image(IMAGEL);

        # storage for input options
integer*2 inopt(INOPTL,INMAX);

        # pointer
integer*2 imagep        # next free entry in image
integer*2 inoptp        # next free entry in inopt

        # next line to write on display
integer*2 lincnt;

        # display mode
integer*2 mode;

        # function code
integer*2 ifunct;

integer*2 idold(10);    # menuid to find in directory
integer*2 idnew(10);    # menuid to store in directory

        # menu parameter
integer*2 nblock;       # number of text/question blocks
integer*2 nbutt;        # number of buttons for a block
integer*2 iloc;         # location of buttons in a block
integer*2 isize;        # size of buttons
integer*2 mentyp;       # menu type
integer*2 ipredr;       # record number of predecessor menu
integer*2 ispace;       # space lines between text blocks
integer*2 irec,param,ichar;
integer*2 mr(21,50),ptr;        # temporary storage for menuid,rec #

integer insave,irsave,iwsave;   # file descriptor from open
common /blk1/image,imagep,inopt,inoptp;
common /blk2/lincnt;
common /blk3/mode;
common /blk4/menuid,ipredm;
common /blk5/ibuff;
common /blk6/nblock;
common /blk7/nbutt,iloc,isize;
common /blk8/mentyp,ipredr;
common /blk9/ispace;
common /blk11/irsave,iwsave;
common /blk12/idold,idnew;
```

```
common /blk13/mr,ptr;



        # specify logical unit numbers and
        # open menufiles for read/write
        # the old menu file 'menufile' is read and
        # it is written onto a temporary menu file 'menuout'.
        # after all is done, 'menuout' is copied to 'menufile'.
call setfil(LUMR,'menufile ');

        # preset inopt,image,menuid,ipredm,ichar,mr
for(i=1;i<=INMAX;i=i+1)
 {
 for(j=1;j<=INOPTL;j=j+1)
   inopt(j,i)=8240;       # SP.'0'
 }

for(i=1;i<=IMAGEL;i=i+1)
   image(i)=8240;         # SP.'0'

for(i=1;i<=10;i=i+1)
 {
 menuid(i)=8240;          # SP.'0'
 ipredm(i)=8240;          # SP.'0'
 }

ichar=8256;     # SP.'@'
ptr=1;
for(i=1;i<=50;i=i+1)
  {
  for(j=1;j<=21;j=j+1)
    mr(j,i)=0;
  }


        # read function code
read(LUR,1000) ifunct;
1000 format(i3);

if(ifunct==PRINTD) call prdir
else
  {
        # save menu-id
   read(LUR,100) ibuff;
   100 format(80a1);

   i=1;
   while(i<=10 & ibuff(i)!=ichar)
    { menuid(i)=ibuff(i); i=i+1; }

   if(ifunct==PRINTI) call primag
   else if(ifunct==PRINTO) call propt;
```

```
   }

        # close all files
rewind LUMR;

end





#**************************************************************************
#
#                       S E T M E N
#
#**************************************************************************
#
# Initial release -- hn/2 Oct 79
# NAME:
#       setmen(irec)
# CALLING PROGRAM:
#       mcheck
# FUNCTION:
#       prepare store for new and old menuid
#       and set menuid into mr
# OUTPUT PARAMETER:
#       irec -- index of menuid in mr
# INFORMAL INPUT:
#       menuid - menu id
# ALGORITHM:
#       reset temporary id stores
#       copy id's
#       save old and new menuid in mr, return index
#
#
##########################################################################

subroutine setmen(irec);


integer*2 iflag,irec;
integer*2 p1,p2;
common /blk4/menuid,ipredm;
integer*2 menuid(10);
integer*2 ipredm(10);

        # store for old and new menuid
common /blk12/idold,idnew;
integer*2 idold(10);
integer*2 idnew(10);
```

```
      # preset temporary storage for id's
for(i=1; i<=10; i=i+1)
  {
  idold(i)=8240;        # SP.'0'
  idnew(i)=8240;
  }

      # store old and new menuid's
  for(i=1;i<=10;i=i+1)
    {
    idold(i)=menuid(i);
    idnew(i)=menuid(i);
    }
      # save menuid in mr
call savemr(0,irec);

return;
end


#***********************************************************************
#
#                          S A V E M R
#
#***********************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       savemr(p1,index)
# FUNCTION:
#       store old and new menuids (idold,idnew) into mr and
#       return the index in mr
#       or store record number and return index
# OUTPUT PARAMETER:
#       index - index in mr
#
########################################################################

subroutine savemr(p1,index);

integer*2 index,p1;
common /blk12/idold,idnew;
integer*2 idold(10), idnew(10);

common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;

if(p1==0){
for(i=1;i<=10;i=i+1)
  {
  mr(i,ptr)=idold(i);
  mr(i+10,ptr)=idnew(i);
  }
}
```

```
else mr(21,ptr)=pl;
index=ptr;
ptr=ptr+1;
return;
end


#**********************************************************************
#
#                        G E T M E N
#
#**********************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       getmen(irec)
# FUNCTION:
#       get the menu record number for a menu
#       whose menuid is in ipredm
# OUTPUT PARAMETER:
#       irec -- record number of first menu record
#
######################################################################

subroutine getmen(irec);
integer*2 irec;

common /blk4/menuid, ipredm;
integer*2 menuid(10), ipredm(10);

common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;

irec=0; # preset
i=1;    # index in mr

while(i<ptr & irec==0)
  {
  j=11; # index for first character of menuid
  while(irec==0 & j<=20)
    {
    if(mr(j,i) != ipredm(j)) break
    else if(j>=20)
      { irec=mr(21,i); break;}
    else j=j+1;
    }
  i=i+1;
  }

return;
end
```

```
#*********************************************************************
#
#                           U P D M R
#
#*********************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       updmr
# FUNCTION:
#       read all menu directory records from menufile and write
#       them to menuout
#       retrieve record numbers for menuid's stord in mr
#       and store them in mr.
# ALGORITHM:
#       open files for read from menufile, write to menuout
#       while there are more directory records
#         read a record
#         for all entries in record
#         compare it with mr entry
#         if found store record number
#         update directory record (new menuid & rec number)
#         write record to menuout
#
######################################################################

subroutine updmr;

common /blk13/mr,ptr;
integer*2 mr(21,50), ptr;
integer*2 p1,p2,p3;

integer*2 rbuff(544), dir(12,45), p1,p2,p3;
equivalence (rbuff(1),dir(1,1));

        # close and open files
call coffwd;
if(ierror(104) != 0) goto 9999; # if END-OF-FILE
icount=ptr-1;   # number of entries in mr

for(i=1;i<=NDREC;i=i+1) # for all directory records
  {
  read(LUMR,100) (rbuff(k),k=1,544);
  100 format(544a1);
  if(icount==0) next;   # done no more entries to check
 for(k=1;k<=45;k=k+1)
  {
  dir(11,k)=dir(11,k)-8192;
  dir(12,k)=dir(12,k)-8192;     # sub SP
  }

  for(j=1;j<=45;j=j+1)  # for all record entries
    {
```

```
    for(n=1;n<ptr;n=n+1)        # for all mr entries
      {
       p1=mr(21,n);
       if(p1!=0 & mr(11,n)==0) {
        call intasc(p1,p2,p3);
        if(p2==dir(11,j) & p3==dir(12,j))
          { for(11=1;11<=10;11=11+1)
               mr(11+10,n)=dir(11,j);
                icount=icount-1;
          }
         }
       else {
      for(1=1;1<=10;1=1+1)      # for all characters of menu-id
       {
       if(p1!=0) break;  # if record number found
       if(mr(1,n)!= dir(1,j)) break  #char dont match
       else if(1==10)     # menuid found
         {
          for(11=1;11<=10;11=11+1)  # copy new menuid
            dir(11,j)=mr(11+10,n);
          p3=NDREC+1+((i-1)*45+j-1)*(NIREC+NOREC);  # rec number
          mr(21,n)=p3;
          call intasc(p3,p1,p2);        # convert into ASCII coded
          dir(11,j)=p1; dir(12,j)=p2;
          icount=icount-1;
          }
        }
       }
      }
    }
  }

         # stop if not all menu id's found
if(icount>0) {write(6,19);19 format('Menu id not found--stop');stop;}
9999 continue;
return;
end
#**********************************************************************
#
#                P R I M A G
#
#**********************************************************************
#
# Initial release -- hn/1 november 79
# NAME:
#       primag
# FUNCTION:
#       print the image of a requested menu
#
######################################################################

subroutine primag;
```

```
common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;
integer*2 dbuff(544),p1,p2,irec;


        # set menuid into mr
call setmen(irec);
        # get menu record number from directory
call updmr;
irec=mr(21,irec);

        # dummy read over all preceeding records
i=1+NDREC;
while(i<irec)
  {
  read(LUMR,100) (dbuff(k),k=1,544);
  100 format(544a1);
  i=i+1;
  }

        # read and print up to four image records
for(i=1;i<=4;i=i+1)
  {
  read(LUMR,100) (dbuff(k),k=1,544);
  p2=1;         # pointer in record,start of command
  for(p1=1;p1<=540;p1=p1+1)
    {
    if(dbuff(p1)==8228 | dbuff(p1)==8316)  # CR LF or ENDOFI
      {
      p1=p1-1;
      if(p1>=p2)        # do not print successive cr/lf
      write(LUW,100) (dbuff(l),l=p2,p1);        # write command
      p1=p1+1; p2=p1+1;
      }
    if(dbuff(p1)==ENDOFI) break; # read next record if any
    if(dbuff(p1)==ENDOFR)
        { i=5; break; }         # stop, done
    }
  }

return;
end




#**********************************************************************
#
#               P R O P T
#
#**********************************************************************
#
# Initial release -- hn/2 november 79
# NAME:
```

```
#       propt
# FUNCTION:
#       print all input options for a menu specified by
#       menu-id in menuid
#
####################################################################

subroutine propt;

common /blk4/menuid,ipredm;
integer*2 menuid(10),ipredm(10);
common /blk1/image,imagep,inopt,inoptp;
integer*2 image(IMAGEL),imagep,inopt(INOPTL,INMAX),inoptp;
integer*2 p1,p2,p3,p4,p5,irec,mentyp,flg;
integer*2 p(5), menu(10),  dbuff(544);
integer*2 run,rp;
common /blk13/mr,ptr;
integer*2 mr(21,50),ptr;
integer int;
equivalence (inopt(1,1),dbuff(1));

call setmen(irec);    # set menu
call updmr;
l=mr(21,irec)+4;
ptr=l;
for(i=1;i<=50;i=i+1)
{
 for(j=1;j<=21;j=j+1)
   mr(j,i)=0;
}
        # dummy read over all preceding records
i=1+NDREC;
while(i<l)
  { read(LUMR,100) (dbuff(k),k=1,544);
    100 format(544a1);
    i=i+1;
  }

        # read both records
read(LUMR,200) (dbuff(k),k=1,272);
read(LUMR,200) (dbuff(k),k=271,542);
200 format(272a2);

run=1;
rp=1;
if(l<=29) p4=36 # start menu
   else p4=29;
int=inopt(3,p4);                # number of input options
p1=int/256; p2=mod(int,256); call ascint(p1,p2,p5);

lr=l-4;
write(LUW,1) (menuid(lll),lll=1,10),lr;
1 format(//'Input options for menu:   ',10a1/,'first record:',i3/);
write(LUW,2) p5; 2 format('number of input options:',i3);
```

```
int=inopt(2,p4);
pl=int/256; p2=mod(int,256); call ascint(pl,p2,p3);
write(LUW,3) p3; 3 format('menu type:',i3);

p2=inopt(1,p4)/256;
int=inopt(1,p4);
p3=mod(int,256);
call ascint(p2,p3,pl);
call savemr(pl,irec);

while(run<=2) {
if(run==2)
  {
  write(LUW,4) (mr(l,rp),l=11,20);
  4 format('predecessor menu: ',10al);
  rp=rp+l;
  }

        # print entries of each input option
for(p4=1; p4<=p5; p4=p4+1)
{
        # convert coordinates and action code into integers
for(l=1;l<=5;l=l+1)
  {
  pl=inopt(l+6,p4)/256; p2=mod(inopt(l+6,p4),256);
  call ascint(pl,p2,p(l));
  }

if(run==2) {
write(LUW,6) p4; 6 format(//'input option',i3);
write(LUW,7) inopt(l,p4); 7 format('label: ',al);
write(LUW,8) (inopt(l,p4),l=2,6);8 format('log command: ',5a2);


write(LUW,9) p(l); 9 format('button left collumn:',i3);
write(LUW,10) p(3);10 format('button right collumn:',i3);
write(LUW,11) p(4);11 format('button upper row:',i3);
write(LUW,12) p(2);12 format('button lower row:',i3);
write(LUW,13) p(5);13 format('action code:',i3);
 }

if(p(5)==1)                # read menu
  {
  pl=inopt(12,p4)/256; p2=mod(inopt(12,p4),256);
  call ascint(pl,p2,p3);
  if(run==1) call savemr(p3,irec)
  else {
write(LUW,14) (mr(l,rp),l=11,20);14 format('successor menu: ',10al);
  rp=rp+l;}
  }
else if(p(5)==2 | p(5)==4)      # return character
  {
  if(run==2)
  write(LUW,15) inopt(12,p4); 15 format('character to return: ',al);
```

```
  }
else    # return integer
  {
  if(run==2) {
  pl=inopt(12,p4)/256; p2=mod(inopt(12,p4),256);
  call ascint(pl,p2,p3);
  if(p(5)==3)           # integer
    {
    write(LUW,16) p3; 16 format('integer to return:',i3);
    }
  else if(p(5)==14)
    {
    write(LUW,17) p3; 17 format('new task id:',i3);
    }
  else; # do not know yet
  }
  }
}
rp=1; run=run+1; if(run<=2) call updnr; }

return;
end




#***********************************************************************
#
#                        P R D I R
#
#***********************************************************************
#
# Initial release -- hn/3 november 79
# NAME:
#       prdir
# FUNCTION:
#       print the contence of all directory records
#
##########################################################################

subroutine prdir;

integer*2 dbuff(544), im(12,45), p(45);
integer*2 pl,p2,flag;
equivalence (dbuff(1),im(1,1));


write(LUW,1); 1 format(//'These are the directory records'//);
        # print all records
for(i=1;i<=NDREC;i=i+1)
{
flag=0;
read(LUMR,100) (dbuff(k),k=1,544);
100 format(544a1);
```

```
write(LUW,2) i; 2 format(/'record number:',i3);

for(k=1;k<=45;k=k+1)
  {
  p1=im(11,k)-8192; p2=im(12,k)-8192;
  call ascint(p1,p2,p(k));
  if(p(k)==248)  p(k)=0;
  if(im(1,k)!=8240) flag=1;
  }
        # write menuid's and record numbers
if(flag!=0) {
for(k=1;k<=43;k=k+3)
  {
  i1=k; i2=k+2; # limits for 3 entries
  write(LUW,3) (((im(n,l),n=1,10),p(l)),l=i1,i2);
  3 format(3(10a1,i3,2x));
  }
flag=0; }
else
write(LUW,77); 77 format('There are no entries');
}
return;
end



#*********************************************************************
#
#               3.1   C O F F W D
#
#*********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#       coffwd
# FUNCTION:
#       close menufile
#       open menufile
#       read from menufile
#
# ALGORITHM:
#       close file
#       rewind file
#       connect logical unit number and file
#
######################################################################

subroutine coffwd;

common /blk11/irsave,iwsave;
integer irsave,iwsave;

rewind LUMR;
```

```
call setfil(LUMR,'menufile ');


return;
end


#*********************************************************************
#
#               I N T A S C
#
#*********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#       intasc(inp,out1,out2)
# FUNCTION:
#       Conversion of an integer (0-900) into two integer*2 ASCII codes
# ALGORITHM:
#       out1= inp/30 +40 (range ASCII 40-70)
#       out2= mod(inp,30) +40 (range ASCII 40-69)
#
#####################################################################

subroutine intasc(inp,out1,out2);

integer*2 inp,out1,out2;
integer num1,num2;

num1=inp;
out1=inp/30+40;
num2=mod(num1,30);
out2=num2+40;
return;
end



#*********************************************************************
#
#               A S C I N T
#
#*********************************************************************
#
# Initial release -- hn/31 october 79
# NAME:
#       ascint(in1,in2,outp)
# FUNCTION:
#       Conversion of an ASCII coded number (0-900) into
#       an integer*2 value
# ALGORITHM:
```

```
#       outp= (inl-40)*30 + (in2-30)
#
##########################################################################

subroutine ascint(inl,in2,outp);

integer*2 inl,in2,outp;

outp=(inl-40)*30+in2-40;
return;
end
```

PATCH FOR RT-11

```
; PATCH 10 TO OTS
        .TITLE  $FCHNL
        .IDENT  /03/
        .PSECT  OTS$I

S = .
        JSR     PC,PAT10
        NOP
        CMP     R1,R0

PAT10:
        .GLOBL  $NLCHN
        MOV     4(R3),R0
        ADD     #$NLCHN,R0
        CMP     R1,R0
        RTS     PC
        .END
```

LIST OF ABBREVIATIONS

ASCII	- American Standard Code for Information Interchange
BG	- Background
CRT	- Cathode Ray Tube
DEC	- Digital Equipment Corporation
FG	- Foreground
HP	- Hewlett-Packard
I/O	- Input-Output
LED	- Light Emitting Diode
NAVCOMSTA	- Naval Communication Station
SATCOM	- Satellite Communications
SB	- Status Block
SSA	- SATCOM Signal Analyzer
PB	- Parameter Block

LIST OF REFERENCES

Bell Telephone Laboratories, Inc., UNIX Programmer's Manual, Sixth Edition, May 1975

Bell Telephone Laboratories, Inc., RATFOR - A Preprocessor for a Rational Fortran

Carroll Manufacturing Co., Touch Input System for Hewlett-Packard 2640 Series Terminal, May 1979.

Digital Equipment Corp., PDP-11 FORTRAN Language Reference Manual, Publication Number DEC-11-LFLRA-C-D, June 1977.

Digital Equipment Corp., RT-11 Advanced Programmer's Guide, Publication Number AA-5280B-TC, November 1978.

Digital Equipment Corp., RT-11 Documentation Directory, Publication Number AA-5285C-TC, March 1978.

Digital Equipment Corp., Introduction to RT-11, Publication Number DEC-11-ORITA-A-D,DN1, March 1978.

Digital Equipment Corp., RT-11 System Message Manual, Publication Number AA-5284B-TC, March 1978.

Digital Equipment Corp., RT-11 System Release Notes, Publication Number AA-5286B-TC, September 1978.

Digital Equipment Corp., RT-11 System User's Guide, Publication Number DEC-11-ORGDA-A-D,DN1, March 1978.

Digital Equipment Corp., RT-11/RSTS/E FORTRAN IV User's Guide, Publication Number DEC-11-LRRUB-A-D, September 1977.

Hewlett-Packard Co., 2641A, 2645A, 2645S/N Display Station Reference Manual, Publication Number 02645-90005, November 1978.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Professor John E. Ohlson, Code 620l Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	6
5.	LCDR Frank Burkhead, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Marineamt - Al - 2940 Wilhelmshaven Federal Republic of Germany	1
7.	Dokumentationszentrale der Bundeswehr (See) Friedrich-Ebert-Allee 34 5300 Bonn Federal Republic of Germany	1
8.	Kapitaenleutnant Heinz-Joachim Niemann Kommando Marinefuehrungssysteme Wibbelhofstrasse 3 2940 Wilhelmshaven Federal Republic of Germany	1

7

l

Thesis 187307
N5823 Niemann
c.1 Background monitor
for the SATCOM signal
analyzer.