SAND94-2143
Unlimited Release
Printed August 1994

Distribution
Category UC-706

Tonopah Test Range EGS Graphics Tracking Display System - HP370

Richard H. Meyer
Experimental Structural Dynamics Department

Kenneth C. Bauhs
Mechanical & Climatic Testing Department

Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

This report describes the HP370 component of the Enhanced Graphics System (EGS) used at Tonopah Test Range (TTR). Selected Radar data is fed into the computer systems and the resulting tracking symbols are displayed on high-resolution video monitors in real time. These tracking symbols overlay background maps and are used for monitoring/controlling various flight vehicles. This report discusses both the operational aspects and the internal configuration of the HP370 Workstation portion of the EGS system.

MASTER

DISTRIBUTION STATEMENT A

Acknowledgments

The authors gratefully acknowledge the technical guidance and cooperation of:

Sandia Personnel

D. F. Anderson, 2719

W. D. Love, 2719-1

S. L. Stronach, 2753

P. Bortniak, 5845

R. J. Rudolph, 2663

EGS Project Leader

Inter-Computer Communications

PC Control Panel

Micro-Vax

Micro-Vax

L. W. Lathrop, 2719

D. F. Anderson, 2719

Global and Range Coordinate algorithms

Global and Range Coordinate algorithms

DISCLAIMER

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

DISCLAIMER

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

Contents

1.0 General Description	1-1
1.1 Overview Enhanced Graphics System	1-1
1.2 Graphics Workstation Functions	1-1
1.2.1 EGS Master Function	1-1
1.2.2 Command Decoding Function	1-5
1.2.3 Graphics Function	1-5
1.2.3.1 Graphics Real-Time Functions	1-5
1.2.3.2 Graphics Playback Functions	1-5
1.2.3.3 Graphics Setup Functions	1-5
1.2.4 Input/Output Function	1-5
1.2.5 Utility Functions	1-5
TTR Coordinate Transforms	
Map Conversion/Editing/Setup	
Diagnostic	
1.3 HP375 Configuration	1-6
1.3.1 Hardware	1-6
1.3.2 Software	1-7
1.4 TTR Supported Test Types	1-8
1.4.1 Air Drop Tests	1-8
1.4.2 Flight Profile Tests	1-9
1.4.3 Balloon Tracking Tests (Wind Run)	1-10
1.5 Communication Formats	1-10
1.5.1 MicroVax II to HP370 Communication Interface	1-10
1.5.2 HP370 to MicroVax II Communication Interface	1-10
1.5.3 HP370 Command Process to Display Process Communications	1-11
1.6 View/Segment Database Format	1-11
1.6.1 Starbase Elements	1-12
1.6.2 Starbase Segments	1-12
1.6.3 Starbase Pruning and Culling	1-12
1.6.4 Starbase Picking (Locator)	1-12
1.6.5 Starbase Filters (Invisibility)	1-13
1.6.6 Starbase Application-Data	1-13
1.6.7 Saving/Restoring Map Database Files	1-13
1.6.8 EGS Views and Segments	1-14
1.7 TTR Coordinate System (NAD27)	1-14
1.7.1 Range Coordinate System	1-14
1.7.2 Range Zero Reference Point	1-15
1.7.3 Master (Earth-Centered) Coordinate System	1-15
1.7.4 Geodetic Coordinate System	1-15
1.8 Miscellaneous	1-15

Contents (Cont.)

1.8.1 Windows.....	1-15
A, B, Cmd, REF, & AER	
Alpha-Numeric window	
Font Manager	
forms (frmlib1.c)	
1.8.2 Plot Symbols.....	1-16
1.8.3 Tracking Objects (Radars).....	1-17
2.0 Operational Modes	2-1
2.1 Graphics Real-Time (GRTP) Mode	2-1
2.1.1 Purpose	2-1
2.1.2 Required Processes	2-1
EGS Master	
Command Interpretation and Control	
Input/Output	
Graphics (Map) Display	
Shared Memory	
2.1.3 Screen Windows	2-1
2.1.3.1 A and B Windows	2-3
2.1.3.2 Command Window	2-3
2.1.3.3 AlphaNumeric Window	2-4
2.1.3.4 AER and REF Windows	2-4
2.1.4 Input Devices	2-5
2.1.4.1 Keyboard (Function Keys).....	2-5
2.1.4.2 Mouse.....	2-6
2.1.5 Master Process.....	2-7
2.1.5.1 Startup Files	2-7
2.1.5.2 Child Processes	2-7
2.1.5.3 X11 Setup Process	2-8
2.1.6 Command Process	2-8
2.1.6.1 Command String	2-8
2.1.6.2 Handshaking.....	2-9
2.1.7 Graphic Display Process (GRTP).....	2-10
2.1.7.1 Initialization	2-10
2.1.7.2 Command Implementation	2-12
2.1.7.3 Screen Refresh	2-15
2.1.7.4 Starbase Events	2-16
2.1.7.5 Radar Symbol Plotting	2-17
2.1.7.6 AER Function	2-17
2.1.7.7 REF and Line Draw Functions.....	2-18
2.1.7.8 AlphaNumeric Data Display	2-20
2.1.7.9 Wind Run Function.....	2-21
2.2 Setup Mode.....	2-23
2.2.1 Purpose	2-23

Contents (Cont.)

2.2.2	Required Processes.....	2-23
	EGS Master	
	Command Interpretation and Control	
	Input/Output	
	X11 Setup	
	Shared Memory	
2.2.3	X11 Windows (Clients).....	2-26
	2.2.3.1 Standard X11 Window Operations	2-26
	2.2.3.2 Sandia Client Format	2-26
2.2.4	Test Project Setup (START)	2-28
	2.2.4.1 Test Information.....	2-28
	2.2.4.2 Startup File ("STADEF").....	2-28
	2.2.4.3 Project Menu.....	2-31
2.2.5	Window Configuration (WINDOW).....	2-32
	2.2.5.1 Window Coordinates.....	2-34
	2.2.5.2 Window Menu Values	2-34
	2.2.5.3 Window Default File ("WINDEF")	2-38
2.2.6	Radar Configuration (RADAR)	2-38
	2.2.6.1 Radar Selection (A, B, and all)	2-38
	2.2.6.2 Radar Plot Menu Values	2-39
	2.2.6.3 Radar Default File ("RADDEF").....	2-41
2.2.7	View Configuration (VIEW).....	2-41
	2.2.7.1 Graphics Windows (A and B).....	2-41
	2.2.7.2 View Menu Values	2-42
	2.2.7.3 View Default File ("VEWDEF")	2-46
2.2.8	Segment/View Editing.....	2-46
	2.2.8.1 EGS Starbase File Types.....	2-47
	2.2.8.2 General View/Segment Procedure	2-48
	2.2.8.3 Segment Editor (SHOW_SEG).....	2-50
	2.2.8.3.1 SHOW_SEG Menu Values	2-52
	2.2.8.3.2 SHOW_SEG Functions.....	2-53
	2.2.8.4 Build View Editor (BUILD)	2-54
	2.2.8.4.1 BUILD Menu Values	2-54
	2.2.8.4.2 BUILD Functions.....	2-57
2.2.9	WayPoint Editor (WAYPOINT)	2-58
	2.2.9.1 WayPoint Text Files	2-58
	2.2.9.2 WayPoint Menu Values	2-59
	2.2.9.3 WayPoint Functions.....	2-63
2.2.10	Air Drop Editor (TCALL).....	2-67
	2.2.10.1 TCALL Menu Values	2-67
	2.2.10.2 TCALL Functions.....	2-71
3.0	Utility Programs.....	3-1
3.1	Map (View) Conversion	3-1
3.1.1	Line-by-Line Editors	3-1
3.1.1.1	Segment Converter (ConvMap)	3-1

Contents (Cont.)

3.1.1.2 View Builder (ViewBld).....	3-3
3.1.2 CAD Graphical Editor (me10).....	3-5
3.2 Diagnostics	3-6
3.2.1 Simulator (Target_sim)	3-6
3.2.2 Target Shared Memory Display (Target_dsp).....	3-10
3.2.3 Command Shared Memory Display (Commnd_dsp)	3-12
3.2.4 Print Display List (PrtDLst)	3-14
3.2.5 TTR Coordinates	3-15
4.0 EGS Libraries	4-1
4.1 Segment File (sfile.h)	4-5
4.2 ASCII File (asclib.h).....	4-20
4.3 EGS (egslib.h)	4-31
4.3.1 Starbase Utility	4-31
4.3.2 Form Utility	4-75
4.4 TTR Coordinate Conversion (ttrcoord.h).....	4-80
Appendix A — HP375 Graphics System (hp4) Hardware Specifications.....	A-1
Appendix B — Target Data Block - uVax to HP370 Data and Command	B-1
Appendix C — Communication Data Block - Command to Display Procedures	C-1
Appendix D — Starbase/Display List General Map Format	D-1
Appendix E — EGS Database File Formats.....	E-1
Appendix F — EGS/TTR Coordinate System (NAD27) Equations	F-1
Appendix G — Building Segment Files (*.sfl) from Tektronix Type Files (*.com).....	G-1
Appendix H — Building Segment Files (*.sfl) from Starbase/Display List Type Files (*.asc)	H-1
Appendix J — EGS Constants and Prototypes (Include Files).....	J-1
Appendix K — EGS/WayPoints Coordinate System Equations.....	K-1
Appendix L — 'Egs_master' and 'Command_proc' Flow Charts	L-1
Appendix M — 'Graphics_proc' Flow Charts.....	M-1
Appendix N — EGS/TTR Save/Restore/Transfer Scripts	N-1

Figures

1.1.1	Overall EGS Block Diagram	1-2
1.1.2	EGS System in TTR Control Room	1-3
1.2	HP370 Software Processes and Intercommunication	1-4
2.1.1	GRTP Software Processes and Intercommunication	2-2
2.2.1	Setup Software Processes and Intercommunication	2-24
2.2.2	X11 Setup Mode Block Diagram	2-25
2.2.3	Sandia X Window Client Format	2-27
2.2.4.1	START Project Client Format	2-29
2.2.4.2	START (Startup) Files	2-30
2.2.4.3	EGS Project Directory Hierarchy	2-33
2.2.5.1	EGS Starbase Viewports	2-35
2.2.5.2	WINDOW - Graphics Window Editor	2-36
2.2.5.3	User WINDOW Selectable Parameters	2-37
2.2.6.1	RADAR - Radar Editor	2-39
2.2.6.2	User RADAR Selectable Parameters	2-40
2.2.7.1	VIEW - View Editor	2-43
2.2.7.2	User VIEW Selectable Parameters	2-44
2.2.8	HP EGS Views and Segments	2-47
2.2.8.1	Building EGS Segments and Views	2-49
2.2.8.2	SHOW SEG - Segment Display Editor	2-50
2.2.8.3	X11 Display "SegWin" Window	2-51
2.2.8.4	BUILD - View Build Editor	2-55
2.2.8.5	X11 Display "ViewWin" Window	2-56
2.2.9.1	'd13tra.txt' WayPoint Text File	2-60
2.2.9.2	WAYPOINTS - WayPoints Calculator	2-62
2.2.9.3	'waypts.asc' Segment File - Arc and Line Segments	2-64
2.2.9.4	'marker.asc' - Circular Markers at Each WayPoint	2-65
2.2.9.5	'annotate.asc' - Text WayPoint Numbers	2-66
2.2.10.1	TCALL - Segment Maker Editor	2-69
2.2.10.2	Example Printer Output from TCALL	2-72
2.2.10.3	GENCAL Segment File - Course Line	2-73
2.2.10.4	HVSY Segment File - Altitude vs. Time	2-74
3.1.1	Building EGS Segments and Views	3-2
3.2.1.1	HP370 Target Simulation ("Target_sim") and Intercommunication	3-7
3.2.1.2	Target Simulator Radar Vector Pattern	3-9
3.2.2	Target Shared Memory (Target_dsp) and Intercommunication	3-11
3.2.3	Command Shared Memory (Commnd_dsp) and Intercommunication	3-13

Figures (Cont.)

Appendices

D.1	Graphics Real-Time Program - Simplified Display Loop	D-3
E.1	Building EGS Segments and Views	E-3
G.1	Tektronix 4081 Viewports.....	G-7
G.2	Building a Segment file (*.sfl) using the Tektronix text files (*.com)	G-8
G.3	'Show_Seg' Setup Window	G-9
G.4	'Build' Setup Window	G-10
H.1	Building a View file (*.vfl) using the Starbase text files (*.asc)	H-5
H.2	'Show_Seg' Setup Window	H-7
H.3	'Build' Setup Window	H-8
H.4	EGS Starbase Viewports	H-10
H.5	EGS Starbase Colors	H-22
H.6	EGS Starbase Marker Types	H-23
H.7	EGS Starbase Line Types.....	H-24
K.1	WayPoint Arc Drawing Algorithm.....	K-5
L.1	'Egs_master' Flow Chart	L-3
L.2	'set_default_values' Flow Chart.....	L-5
L.3	'Init_Shm_Memory' Flow Chart	L-6
L.4	'Command_proc' Flow Chart.....	L-7
L.5	'check_cmd' Flow Chart.....	L-8
M.1	'Graphics_proc' Flow Chart	M-3
M.2	'Set_Window_Parms' Flow Chart.....	M-11
M.3	'Refresh_Screen' Flow Chart	M-13
M.4	'Draw_Static_windows' Flow Chart	M-19
M.5	'Check_Events' Flow Chart.....	M-20
M.6	'Time_Search_Value' Flow Chart.....	M-26
M.7	'Plot_Radar_Sym' Flow Chart	M-27
M.8	'Display_AER' Flow Chart	M-32
M.9	'Display_REF' Flow Chart.....	M-35
M.10	'Draw_Line' Flow Chart	M-36
M.11	'form3' Flow Chart.....	M-42
M.12	'form3_fill' Flow Chart	M-45
M.13	'Wind_Run' Flow Chart	M-48
N.1	EGS Directory Hierarchy	N-3

1.0 GENERAL DESCRIPTION

1.1 Overview Enhanced Graphics System

This document describes the new HP370 component of the Enhanced Graphics System (EGS) that is used at Tonopah Test Range (TTR). The new EGS system replaces an older system that consists of a Perkin-Elmer computer and a Tektronix 4081 graphics workstation.

The purpose and use of the Enhanced Graphics System is further explained in the memo 'Enhanced Graphics Support Requirements' by D. F. Anderson (2712), dated June 26, 1991. Included, for reference, is a block diagram (Figure 1.1.1) of the overall EGS system. Figure 1.1.2 is a photograph of the EGS system as it is laid out in the control room at TTR.

The main purpose of this paper is to describe the Graphics portion of this EGS system, which consists of the HP370 workstation and its associated peripherals.

1.2 Graphics Workstation Functions

The HP370 workstation serves several functions in the EGS system. It not only displays the graphic maps, but also serves as an operator keyboard input for the system.

The main operator interface to the EGS system is through an MS-DOS based companion system, which functions as the control panel. This control panel, with all switch functions, is documented in the 'Control Panel Switch Functions, Enhanced Graphics System' by D. F. Anderson (2712), dated August 9, 1991.

When the operator depresses a switch, a command string is generated and sent to the MicroVax II computer. The MicroVax command processor checks this string and routes it to its final destination. If intended for the Graphics system, the command string is included with the data block (queue) sent to the HP370. The formats for the command string and the data block are included in Appendix B.

Internal to the HP370, four processes (programs) must execute simultaneously to accomplish the required tasks (Figure 1.2). They are the Master process, the Command process, the Graphics process, and the Input/Output process. Each process communicates (passes data) to the other processes by means of Shared Memory blocks. Two Shared Memory blocks exist in the EGS system, Command and Reply/Target. The formats are shown in Appendices B and C.

1.2.1 EGS Master Function

The HP370 EGS Master process is the overall controller for the Command and the Graphics processes. It creates the Command Memory block. Then it starts (spawns) the Command and the Graphics procedures as 'child' processes. The EGS Master process monitors the status of each 'child' and performs system cleanup when the EGS system is halted.

Which Graphics process is started depends on the current value of the 'Operating Mode' parameter in Command Shared memory.

Figure 1.1.1. Overall EGS Block Diagram

Figure 1.1.2. EGS System in TTR Control Room

Figure 1.2. HP370 Software Processes and Intercommunication

1.2.2 Command Decoding Function

The HP370 Command process must decode the command string and route the required data to the final process(es). These final destinations may be the Real-Time Graphic Display procedure, the Graphics Setup procedure, the EGS Master procedure, and/or back to the MicroVax.

1.2.3 Graphics Function

1.2.3.1 Graphics Real-Time Functions

The Graphics Real-Time Process (GRTP) displays operator selected maps in two windows (A and B) on the display screen. The operator can change views and segments by pressing switches on the PC control panel.

In each window, the track of up to five radars may be plotted in real time (Maximum update rate - 1/10 second). This plotting data is read from the data block (Target shared memory) received from the MicroVax computer.

In addition, a third window displays selected alphanumeric data in real time (1/second). The type of data displayed in this window will be fixed at setup time, but usually includes time, range, heading, and altitude information. Certain data values can be changed at the operator control panel (A and B references, Radars, etc.) and some calculated values for vectors will be displayed as required.

1.2.3.2 Graphics Playback Functions

The Graphics Playback process is identical to the Real-Time Process (GRTP) because it just accepts data from the MicroVax and the Control Panel as it would in real time.

1.2.3.3 Graphics Setup Functions

The Graphics Setup process is a separate set of procedures that execute under the X11/Motif window system. The EGS Graphics system default and setup parameters may be changed using these routines. User-changeable parameters include Radar symbols, Window size/location, View (A and B) composition, and Test ID information. In addition, the user may create and/or modify new segments.

1.2.4 Input/Output Function

The communication with the MicroVax computer is handled as a 16-bit bidirectional parallel transfer. In software this data transfer is controlled by a Unix daemon process called 'pollgpio'. This daemon process is started when the HP370 is first turned on (booted) and is present whenever the machine is operating. The daemon process is responsible for initiating the 'Target/Reply' Shared Memory block, loading/reading data in the block, and controlling the hardware handshaking with the MicroVax computer.

1.2.5 Utility Functions

Several off-line functions are available with the EGS system. These utilities are not available in the integrated EGS environment but may be called as individual programs. They were written in order to get the system functional and keep it running.

1.0 General Description

These utility functions fall into the following categories:

1. TTR coordinate transformations

'ttrxyz' - convert NAD27 latitude, longitude, and height to TTR range xyz

'ttrxyz84' - convert WGS84 latitude, longitude, and height to TTR range xyz

'ttrllh' - convert TTR range xyz to NAD27 latitude, longitude, and height

'ttrllh84' - convert TTR range xyz to WGS84 latitude, longitude, and height

2. 'ConvMap' - ['ShowSeg' in integrated X11 environment]

Map Conversion (Old system maps to new segment format)

Segment Setup/Drawing (Enter new segments)

Segment Editing (Change existing segments)

3. 'BldView' - ['Build' in integrated X11 environment]

View Setup (Combine Segments into Views)

View Editing (Change existing Views)

4. 'PrtDLst'

Diagnostic (Examine contents of a segment [*.sfl] or view [*.vfl] file)

5. 'Target_sim'

Diagnostic (Simulate input from MicroVax to Target shared memory)

6. 'Target_dsp' - substitute for 'Graphics_proc'

Diagnostic (Display the contents of Target shared memory in real time)

7. 'Commnd_dsp' - substitute for 'Graphics_proc'

Diagnostic (Display the contents of Command shared memory in real time)

1.3 HP375 Configuration

1.3.1 Hardware

The Graphics Display workstation consists of an HP375CH computer system with associated peripherals.

It contains a 32-bit MC68030 processor with an MC68882 floating point coprocessor. Both of these processors operate with a 50 MHz clock to provide 13.3 MIPS operation.

The system contains 16 Mbytes of Error Checking and Correcting (ECC) Random Access Memory (RAM) with Direct Memory Access (DMA).

The following Input/Output (I/O) interfaces are included with the system:

- RGB monitor output
- RS232 interface
- HP-HIL interface
- Parallel interface
- HP-IB interface
- SCSI interface
- GPIO interface

An Ethernet (IEEE 802.3) Thin-LAN network interface is included with the system.

The workstation contains a 663-Mbyte hard disk for program and data storage. Also, a 32-track, 133-Mbyte cartridge tape may be used for archival storage and transferring data between systems.

The most prominent feature of the HP375 workstation is a high-resolution color monitor. It has a resolution of 1280 x 1024 points with 8 image planes and 2 overlay planes.

The RGB video outputs drive a raster scan converter, which produces NTSC video. The standard video is fed to the range video distribution system so that the HP370 screen display is available at other mission support consoles. Additionally, the RGB outputs, without conversion, drive a large RGB display at the Test Director console to provide high-resolution graphics at that position.

A keyboard, graphics tablet, and mouse provide input to the workstation. They operate over the HP-HIL interface.

A Paintjet color printer provides both program listings and high-quality graphics hard copy.

Detailed listings of all the hardware components are included in Appendix A.

1.3.2 Software

The operating system on the HP370/HP375 workstation is HP-UX, version 9.0 (UNIX).

Program support is provided for the 'C', Fortran, and Assembly languages. However, all programs for the EGS system are written in the 'C' language.

Network support is provided for a wide range of services. These services include file transfer, remote file access, remote terminals (logon), and many others.

Complete X-windows (X11, version A.05.00) with OSF/Motif system installation allows many different independent applications to run on the monitor, each in its own window (environment). It also allows foreground and background program operation.

1.0 General Description

All of the real-time graphics (GRTP) applications for the EGS system are written using the 'Starbase' graphics library. This includes the 'Display List' option to form the view database.

The X-window system is turned off during the 'real time' and 'playback' operating modes of the EGS System. Likewise, the 'Starbase' real-time applications are turned off during the 'Setup' operating mode.

1.4 TTR Supported Test Types

At the present time, there are five general categories of tests supported by the EGS graphics control system. The five categories are as follows:

- Air Drops
- Flight Profiles
- Balloon Tracking
- Rocket Launches
- Gun Round Tracking

These test categories appear to be sufficiently different in setup and map configuration that a separate procedure may be required for each.

1.4.1 Air Drop Tests

The Air Drop tests at TTR are characterized by the tracking of multiple, high-speed objects in real time (1/10 second or 1 second update rate).

Each of the objects will be tracked by a separate radar for display purposes. Provisions will be made to allow for up to five simultaneous radar (object) tracks on the display. However, any one of the radars (objects) may be declared as primary (A) for calculation purposes. Also, a secondary (B) radar may be selected for some calculations. These calculated functions are the following (FORM3):

- Track 'A' Location (Time to Target) from Target
- Track 'A' Magnetic Heading, Altitude, and Velocity
- Track 'B' Magnetic Heading, Altitude, and Velocity
- Bearing and Range (Track 'A' to Track 'B')

The 'TCall' setup procedure is used to create an X-Y course line (run-in line) segment with time (to release point) indications. Also, the 'TCall' procedure may be used to create an altitude versus distance-from-target segment. When combined with other TTR range segments, they will form 'Views' in both the X-Y (A View) and Y-H (B View) planes. This provides the range controller a three-dimensional image of the range. The radar tracks are superimposed on these background views.

Each of these Display views (A and B) is referenced to a common target reference point. As the radar-tracked object nears this target, the controller successively zooms the View (A and/or B) toward the target to provide better resolution (detail).

During a test, the heading of certain objects to the target and to each other may be required in order to allow the operator to direct them toward (away from) each other.

As the plane approaches the target, the graphics operator informs the pilot by radio when he is at the precalculated release point. This position, heading, and time information must be selectively provided in text form for the operator (alphanumeric window).

At operator-selected times during each test the position, heading, and time information must be saved on the screen.

Also, at certain critical times the operator may choose to freeze the screens (Displays A and B) and/or make copies of the screen. During these times the tracking display algorithm must keep running, so that when the screen is unfrozen the display will automatically return to normal tracking.

The symbol update rate (Peck Rate) and the length of the symbol trails (Trail Length) must be operator-selected (changeable) from the control panel during a test.

1.4.2 Flight Profile Tests

The Flight Profile tests at TTR are distinguished by the tracking of multiple, high-speed objects in real time (1/10 second or 1 second update rate). The tracked objects must follow a predefined profile (trajectory) and may end with an Air Drop (see Section 1.4.1).

Each of the objects will be tracked by a separate radar for display purposes. Provisions will be made to allow for up to five simultaneous radar (object) tracks on the display. However, any one of the radars (objects) may be declared as primary (A) for calculation purposes. Also, a secondary (B) radar may be selected for some calculations. These calculated functions are the following (FORM3):

- Track 'A' Location (Time to Target) from Target
- Track 'A' Magnetic Heading, Altitude, and Velocity
- Track 'B' Magnetic Heading, Altitude, and Velocity
- Bearing and Range (Track 'A' to Track 'B')

The 'WayPoints' setup procedure is used to create an X-Y flight path segment, with waypoint markers and annotation numbers. When combined with other TTR range segments, they form a 'View' in the X-Y (A View) plane. The radar tracks are superimposed on this view.

During a test, the heading of certain objects to the flight path and to each other may be required in order to allow the operator to direct them toward (away from) each other.

At operator-selected times during each test the position, heading, and time information must be saved on the screen.

Also, at certain critical times the operator may choose to freeze the screens (Displays A and B) and/or make copies of the screen. During these times the tracking display algorithm must keep running, so that when the screen is unfrozen the display will automatically return to normal tracking.

The symbol update rate (Peck Rate) and the length of the symbol trails (Trail Length) must be operator-selected (changeable) from the control panel during a test.

1.4.3 Balloon Tracking Tests (Wind Run)

The Balloon Tracking tests at TTR are used primarily in preparation for other range tests. They are used to generate Wind Profile data before conducting other tests.

Normally, a helium-filled balloon is launched and is tracked by a radar for display purposes. On the EGS graphics display system, this balloon tracking radar must be declared as Radar A for calculation purposes. The Wind data, velocity and direction, is calculated and displayed on the CRT screen and the line printer in real time. At the current time, no Wind data is archived to disk files. In addition, all other calculated functions and displays are also active as defined in the Air Drop section (1.4.1).

The Balloon Tracking procedure is initiated by depressing the 'Wind Run' function key (f4) and is stopped by pressing the 'Stop Wind' function key (f5). Each time the 'Wind Run' key (f4) is pressed, a new zone (wind velocity and direction) is calculated and displayed (REF window and printer).

Each of the Display views (A and B) must be referenced to a common target reference point (weather station) for accurate altitude information.

The symbol update rate (Peck Rate) and the length of the symbol trails (Trail Length) must be operator-selected (changeable) from the control panel during a test.

1.5 Communication Formats

1.5.1 MicroVax II to HP370 Communication Interface

The HP370 Graphics System communicates with the MicroVax to obtain both command and plotting data. This hardware interface is implemented by an HP98622A (GPIO) card at the HP370 side and a DVR11-WA General Purpose DMA interface card on the MicroVax side.

These interfaces provide a 16-bit parallel, bidirectional communication path between the two computers.

On the HP370 computer, the data is read by a background 'daemon' program (pollgpio) that reads the interface card and stores the data values in 'Target' shared memory. It also notifies attached programs that new data has been received. The data is double-buffered and the using procedure has .1 second to retrieve the memory data before it is overwritten.

Software for both sides of this interface was written by W. D. Love (2719). The data may be accessed by macros that are documented on the HP370 by using the 'man' commands.

Included in Appendix B is a listing of the 'Target' shared memory data block received from the MicroVax.

1.5.2 HP370 to MicroVax II Communication Interface

The same interface hardware (GPIO/DVR11-WA) is used to send replies from the HP370 to the MicroVax. These replies are answers to some command query (solicited responses).

On the HP370 computer, the reply data is transmitted by the same 'daemon' program (pollgpio) that acts as a driver to the GPIO interface. The replying program must put the data in the 'Reply' shared memory.

Macros for performing this task are provided by W. D. Love (2719). They are documented on the HP370 by using the 'man' command.

The 'Reply' shared memory data block sent to the MicroVax is included in Appendix B.

1.5.3 HP370 Command Process to Display Process Communications

Four processes are running simultaneously in the HP370 Graphics computer. These are the EGS Master process, the Command process, the I/O Driver process, and the Graphics Display process. In order to function, each process must pass data to the others. These data may represent new command updates from the control panel, initial setup parameters, and/or error information.

The method of passing this information is by a 'Command' shared memory block that each program must attach. This format is further defined in Appendix C. The EGS Master process is responsible for initializing this shared memory, while all processes maintain and use it.

The values in 'Command' shared memory are the current state of the control panel switches, values as read from a setup file, or values input during the 'Setup' mode.

The window parameters are stored in this memory and they define the real-time window sizes and locations. They may be changed by the 'Setup' Graphics Process (WINDOW). In general, windows A, B, and Alpha are used by the Graphics Real-Time process for displaying plot information. Likewise, the Cmd window is a pop-up window that is used by the Real-Time Process to enter a user's response to a Control Panel request. Two extra pop-up windows (AER and REF) are allocated for special uses.

The filenames and switch locations for each view and segment are also stored in this area to facilitate setup (VIEW).

1.6 View/Segment Database Format

The graphics routines used to construct the map segments on the EGS Graphics system are drawn using the 'Starbase graphics' library. In addition, 'Display List' software runs with the graphics software to provide a hierarchically organized graphics database. This view/segment database may be created, edited, displayed, saved to disk, and recovered from disk.

A detailed description of the view/segment formats are included in Appendices D and E.

Appendix D contains an outline (Pseudo Code) of the graphics functions required in the Graphics Real-Time Program (GRTP). This represents the final form, after segment files have been loaded from disk, as is executed by the program. Also, the Notes indicate the type of 'Starbase/Display List' commands that are used to implement each category.

The General Overhead portion of the Pseudo Code must be in the initialization section of any Starbase graphics program. The Main View and each Segment (subN) are loaded during program execution as new 'Views' are requested. Plotting in both the A and B windows follows this general format.

Appendix E describes the four EGS database file formats as they are stored on disk. Note only the 'View' and 'Segment' file formats may be directly loaded by the Graphics Real-Time Program (GRTP).

1.6.1 Starbase Elements

The basic graphics construction unit in the EGS database is an element. An element consists of a Starbase command, a Display List command, and/or application-specific data. In the database, an element pointer is maintained to keep track of which element will be acted on next. If needed, an element may have a 'label' associated with it.

1.6.2 Starbase Segments

The individual elements are grouped together to form 'segments.' These segments may call other segments to form the overall display view. The view is displayed by traversing the segments in sequential order using the element pointers.

Each individual segment in the view database must be assigned a segment number. This segment number is assigned when building the view in a top-down method. This top-down method of assigning segment numbers is required because the most general TTR range segment (main) is entered first. Then more detailed segments are added in a hierarchical form.

The main/overall views will be assigned numbers of 1, 2, ..., 8 corresponding to the 'View' switch numbers. The next layer in the Display List hierarchy will be assigned sequential numbers under the main views (i.e., 11, 12, ..., 18) and correspond to the 'Segment' switches. Any subsequent layered segments will be assigned corresponding sequential numbers (i.e., 111, 112,...), but they cannot be turned off from the control panel.

Each individual Display List segment will have a 'label' associated with it through level two. These labels correspond to the 'View' and 'Segment' numbers respectively.

When a Display List segment calls another segment a subset of the current Starbase state will be saved. Upon return from the called segment, this state is restored.

1.6.3 Starbase Pruning and Culling

The Starbase/Display List library contains provisions for conditionally traversing the map and selectively displaying certain segments. This capability for 'Pruning' and/or 'Culling' the database in real time, together with 'window clipping,' is a very powerful display tool.

Pruning is the process of ignoring the traversal of segments in the display list if they are outside the current clip limits. This speeds up the map graphics display process.

Culling is the process of ignoring the traversal of segments in the display list if they are smaller than a set threshold size. This process also provides the traversal of a particular segment if some detail is larger than a threshold and a different segment if it is less than the threshold.

Neither Pruning nor Culling has been implemented in the current EGS version of software.

1.6.4 Starbase Picking (Locator)

The Starbase Display List has the ability to detect (identify) segments and/or elements of the map database hierarchy. This capability is called picking and is implemented using the HP370 'mouse' as a 'locator/choice' device.

This picking capability is not used in the real-time program (GRTP), but the 'mouse' is used in two functions (AER and Draw Line).

The 'mouse' locator device is an X-Y tracking device that allows X and Y coordinates to be input to the program. Any of the three switches on the 'mouse' perform the 'choice' function by signaling the program when the X and Y coordinate values are valid.

1.6.5 Starbase Filters (Invisibility)

Filters, as defined in Starbase, allow certain segments (or elements) to be selectively traversed or not traversed depending on certain criteria. This process is fast when executing but a fair degree of complexity is added when setting up the database. There are three types of filters defined for Starbase.

The invisibility filter eliminates (does not traverse) graphics primitives that have been defined as invisible and passes all primitives that have been defined as visible.

The highlighting filter allows the highlighting attribute to be selectively turned on/off for certain graphics primitives.

The unpickability filter eliminates certain graphics primitives from being pickable by the 'locator' device (mouse).

The Display List invisibility filter process is implemented for the EGS system and is the procedure by which each level two segment is turned on/off by the 'Segment' switches.

Highlighting and unpickability filters are not used at this time.

1.6.6 Starbase Application-Data

Information, other than graphics data, may be stored in the map database by means of the 'application-data' process.

This data is in alphanumeric form and includes header information for each view or segment. It is always the first element in each segment.

1.6.7 Saving/Restoring Map Database Files

The map files for the HP370 EGS Graphics system may be stored in four formats.

The first format is an 'ASCII' file format (*.asc), which is a direct representation of the Starbase/Display List graphics commands. This text format is in a user-readable form and is intended to be used for debugging and troubleshooting purposes. It is also useful when transferring drawings between graphics systems or creating EGS drawings in a non-Starbase environment.

The second is the Display List 'Segment' format (*.sfl), which is a binary file defined in the Display List library. It can be edited and combined with other segments to form the final view files. A description of this format is included in Appendix E. These files may be loaded into the Graphics Real-Time Program (GRTP), but they would not be filtered (always on).

1.0 General Description

The third format is the 'View' file format (*.vfl), which is a binary file that incorporates the segment hierarchy defined for the EGS system database. It is similar to the segment format, but includes the filter parameters and the Segment calls. These files are loaded into the Graphics Real-Time Program (GRTP) for each 'View' switch entry.

A fourth format is the 'bitmap' format, which is used to denote a frame buffer or memory buffer image. This format is used by the EGS Graphics Real-Time Program to store views of the screen for subsequent 'spooled' output to the printer. Sample 'C' subroutines to store/restore bit map images are included in Appendix E but are not commonly used.

1.6.8 EGS Views and Segments

The Enhanced Graphics System (EGS) control panel has two display windows (A and B). Each display (A or B) window may show one of eight 'Views.' In addition, each 'View' may be further divided into eight 'Segments.' They may consist of the main range layout, the set of radar locations, optical site locations, and many other drawings. They are built, selected, and loaded for each test as determined by the test conductor.

The term 'View' as defined by the EGS control panel is a main/overall segment (*.vfl) in the display list library hierarchy.

The 'Segments' on the EGS control panel are level two (*.sfl) in the display list hierarchy and are turned on/off utilizing the invisibility filter function.

1.7 TTR Coordinate System (NAD27)

The TTR coordinate system is a right-handed Cartesian coordinate system. It may be converted to other global coordinate systems and range survey data may be exchanged with other ranges.

All calculations done in the EGS graphics system are done using range coordinates in meters. However, in the WAYPOINTS editor, provision will be made to allow the entering of EGS segments in either range (x,y,z) or global (latitude, longitude, and height) coordinates.

Also, during Setup, range coordinates may be entered in English (feet), metric (meters), or nautical miles.

In the Graphics Real-Time Program (GRTP), altitude will always be given in feet above mean sea level (msl).

1.7.1 Range Coordinate System

The TTR range coordinate system is a right-handed Cartesian coordinate system with the x-y plane normal to the vertical (+z) axis. The x positive (+x) axis points eastward and the y positive (+y) axis points northward. The units may be in either feet or meters from the range zero point (main lake bed).

The Cartesian (x,y,z) coordinate system may easily be converted to a slant range, azimuth (north), and elevation (+up) system by the equations in Appendix F.

The magnetic declination at TTR is approximately -16.5 degrees from true north. This represents the deviation of magnetic north (compass) azimuth from true north (earth's rotational axis) at Tonopah.

1.7.2 Range Zero Reference Point

The TTR range zero reference point lies 5331.34 feet (1624.992 meters) vertically below the main lake target. This point is assumed to be at mean sea level (msl) and therefore lies on the geoid (equipotential surface).

The difference between this geoid surface and the ellipsoid surface (NAD27) is the geoid undulation. This geoid undulation has a value of -18.2 meters at TTR. At the present time, this value is only used when translating the coordinates to an earth-centered system (Section 1.7.3). It is small compared to the earth translation values and is neglected at this time.

1.7.3 Master (Earth-Centered) Coordinate System

The Master (E,F,G) coordinate system is a right-handed Cartesian coordinate system located at the center of an ellipsoid of reference. At TTR this ellipsoid is the 'Clarke' ellipsoid of 1866 and is also known as the 'North American Datum' (NAD27).

The specific parameters for this datum along with the translation equations are given in Appendix F.

The E-F plane lies in the plane of the equator with the G-axis along the rotational axis of the earth. E is positive in the longitude ($\emptyset$) of 0 degrees (Greenwich England) and the latitude (λ) of 0 degrees (equator). G coincides with the earth's rotational axis and is positive through the North Pole. F lies in the plane of the equator (along with E) and forms a right-handed system (+F east).

1.7.4 Geodetic Coordinate System

The geodetic ($\lambda, \emptyset, h$) coordinate system assumes the point is referenced to the ellipsoid surface (earth-centered). As mentioned in Section 1.7.3, this is the 'Clarke' ellipsoid of 1866 (NAD27). The geodetic latitude (λ) is positive north, and the geodetic longitude ($\emptyset$) is positive east of Greenwich. Height (h) is positive above the ellipsoid surface along the geodetic normal to the surface.

Translation equations between the master (E,F,G) and geodetic systems are given in Appendix F.

1.8 Miscellaneous

1.8.1 Windows

The Graphics Real-Time Program (GRTP) utilizes up to six separate windows to display information on the screen. Each window has a separate function and is a separate entity when defining its graphics attributes (position, size, etc.). The attributes for each window are selected at setup time.

The physical CRT screen is divided, by hardware, into image planes and overlay planes. Both the image and overlay planes seem to reside in the same physical screen space. However, the overlay planes may be made 'transparent' with the image planes underneath the overlay planes. Image plane drawings may contain 256 colors while the overlay plane objects are limited to four colors.

The 'A' window is one of the main plotting windows and is used when plotting objects (radar symbols or views) in X versus Y range coordinates. It may be selectively turned on or off. By default this A window normally resides in the image planes.

1.0 General Description

The 'B' window is the other plotting window and is used when plotting objects (radar symbols or views) in Y versus H(msl) range coordinates. It may be selectively turned on/off and by default resides in the overlay planes.

Text data is displayed in the Alpha-Numeric window. This window cannot be turned off and by default resides in the image planes. It differs from the previous A and B windows because Starbase graphics functions are not used. Instead, the Font Manager library provides a low-level textual interface to the graphics window.

Certain predefined forms are provided in the graphics real-time library to be displayed in this Alpha-Numeric window. The exact form number may be selected by the user at Setup time. Each form consists of two parts, the 'form' background and the 'form' fill parts. The background portion is only executed when the entire screen is refreshed. The fill portion is executed at a 1-per-second update rate to display changing text information. This division of the background from the fill portions saves considerable time in the real-time (GRTP) control loop. Each portion is implemented as a function and changes (form number) are passed (in the main program) as 'pointers' to functions.

The Command window is a pop-up window that is used to display messages and query the operator for responses to questions. This window is normally off and by default resides in the overlay planes. It uses the Font Manager library to display text information.

Another pop-up window is the REFeRence window, which is used to display the current reference point coordinates. The window is normally off and by default resides in the overlay planes. It can be turned on by a control panel switch and uses the Font Manager library functions.

Another function that utilizes this window (REF), to display the X and Y coordinates, is the line draw function. Also, the 'Wind Run' function uses this window (REF) to display zone data. Whichever function first grabs the window uses it until it is finished; other functions are locked out.

The AER window is another pop-up window that is used to display the Azimuth, Elevation, and Range information. This window is normally off and by default resides in the overlay planes. It is turned on by the control panel switches and uses the Font Manager library.

1.8.2 Plot Symbols

The Graphics Real-Time Program (GRTP) uses Starbase markers as plotting symbols. There are 18 default marker types in Starbase and they are shown in Appendix H.

Three plot symbols (markers) are fixed by the real time program. The **A Radar** symbol is an "X" (marker 4) with a fixed color of Red (2). The **B Radar** symbol is always a square_cross "⊞" (marker 8) and is Yellow (3). These two marker colors were selected because they are defined in both the image and overlay planes. The **Trail** symbol is a circle (marker 3) and is the same color as its corresponding plot symbol. The symbol size for Radar symbols is fixed at 1% (.01 - Virtual Coordinates) of the plot window size. The Trail symbol size is fixed at one quarter (.25) of the Radar symbol size.

Each of the other Radar symbols (1 -> 8) has a user-selected marker type, color, and size. By default the marker symbol and Radar number are set to the same value {Radar #1 -> Numeral "1" (marker 10)}. These default values may be changed in the RADAR process (Setup mode). Also, note these additional symbols are only visible when the 'Plot All Radars' function is active.

1.8.3 Tracking Objects (Radars)

Each object to be plotted must be individually tracked by a separate radar system. These radar tracking coordinates are transmitted, by the Tracking Vector Translation System (TVTS), to the Main Control Building (CP). The TVTS reader connected to the MicroVax computer reads these radar data streams. Then the MicroVax reformats the radar tracks and passes the data to the HP370 system for plotting. The current capacity of the TVTS system is a maximum of five radars.

At the MicroVax, a table of radars and their locations is maintained and the operator may select Radar 1 (etc.) to be a specific radar (R36, etc.) track. Also, from the PC Control Panel a radar may be designated as Radar A (or B). The current radar name is usually displayed in the Alphanumeric window (FORM3) for Radars A and B.

In the 'Plot All Radars' mode, the actual radars (objects) that are plotted are determined from this MicroVax table and the corresponding tracking data that is piped across the parallel interface.

1.0 General Description

2.0 OPERATIONAL MODES

2.1 Graphics Real-Time (GRTP) Mode

2.1.1 Purpose

The Graphics Real-Time Program (GRTP) receives coordinate data from the MicroVax computer and displays it on the CRT screen. The display allows the range controller to quickly determine the relative positions of tracked objects in both time and space. This radar tracking data is received via coax cable from the Target Vector Translation System (TVTS) multiplexor(s) located in the Central Computer room, and represents test events immediately as they occur at the range. The usual form of the tracking display is an X-Y plot of radar data overlaid onto maps specially prepared for the test at hand. In addition, an altitude display and selected text information may be shown in separate windows.

2.1.2 Required Processes

Four software processes (programs) must be executing in the HP370 Graphics system when the plotting display is active. The 'Egs_master' process starts the other processes (except 'pollgpio') and coordinates the active processes. 'Egs_master' must always be present. The 'pollgpio' daemon is started at system boot time and controls input/output communications with the MicroVax computer.

The 'Command_proc' process interprets the command string received from the MicroVax/Control Panel and performs the steps required to ensure its execution. In most cases, this implies setting the proper parameter (Command.h) for the 'Graphics_proc' program. At other times, the parameter may be read by the 'Egs_master' program or a system value (Command.h) may be returned to the MicroVax (solicited response).

The 'Graphics_proc' process plots radar tracks on the screen, displays range maps, and implements the majority of the Control Panel functions. The one category of Panel functions that is not implemented is the Setup mode functions, which are described in Section 2.2.

Each of the previously mentioned processes communicates with the other processes by means of a Unix operation called shared memory blocks. This operation is further documented under the 'inter-process communication (ipc)' heading in the Unix literature. These blocks are fixed memory locations (sizes) that are accessible by several different programs.

The exact structure for the 'Target/Reply' shared memory block is described in Appendix B.

Appendix C details the 'Command' shared memory block structure.

2.1.3 Screen Windows

The graphics display functions for the plotting program are implemented using HP Starbase and Display List library functions. The display screen (HP98550a) is physically divided into image planes(8) and overlay planes(2). The screen size is 1280 x 1024 pixels.

Figure 2.1.1. G RTP Software Processes and Intercommunication

Internal to the EGS graphics program (GRTP), the screen is organized into six Starbase windows. Each of these software windows is totally independent from the other windows, except they must coexist on the same physical screen. The user may select the location and size of each window at Setup time (Section 2.2).

2.1.3.1 A and B Windows

Windows A and B are the only two graphics plotting windows used by the Graphics Real-Time Program (GRTP).

The exact size, shape, and location (image or overlay planes) for each of these windows may be changed by the user at Setup time (WINDOW). However, it is the user's responsibility to prevent window overlapping (in the same planes) or undesirable interactions may occur. **Note:** The View segments in the EGS database library were drawn based on a square (Isotropic) window. If the plot window is not a square one, some distortion will result.

During Setup, the user may select which tracking data (x, y, z, h, origin, ref. point) is to be plotted on each window axis (X and Y).

These Setup options allow a great deal of flexibility in screen layout and in what radar data is actually plotted during a test.

A user may select (VIEW) any View (or Segment) in the EGS database for display in either the A or B Windows. Up to eight Views may be selected for the A Window and an additional eight Views may be selected for the B Window. Each View can have up to eight Segments (subviews). Only one View at a time can be displayed in a window and each of its Segments can be switched on or off. The same View file may be used multiple times and each occurrence can have a different offset (pan), scale (zoom), and rotation factor.

The current default setting for the A Window is a square viewport (0.0 -> 1.0 virtual coordinates) in the image(0) planes. The plot values are to plot x (referenced to range origin) on the horizontal window(A) axis and to plot y (origin) on the vertical window(A) axis.

The current default for the B Window is a square viewport (0.0 -> 1.0 virtual coordinates) in the overlay(1) planes. This means the two windows overlap each other, but they do not interfere with each other. The plot values are to plot y (origin) on the horizontal window(B) axis and to plot h-msl (referenced to range mean-sea-level point) on the vertical window(B) axis.

2.1.3.2 Command Window

The Command window is a 'pop-up' text window that is only visible when operator keyboard input is required by the plotting program. This may be in response to a command from the MicroVax (Line Draw, New Ref. Point, Time Search Value, etc.) or a local real-time query (Quit). Another use would be to display error messages or user prompts.

Fast Alpha/Font Manager library routines are used to write text strings to the screen. Several font sizes are available (see page A-2, 'Fast Alpha Font Manager Programmer's Manual') for use with these library functions. The window size is fixed by both the font size and quantity of text displayed in it. If a smaller window is desired, or a different hardware screen is used, the font size may need to be changed.

During the time that the Command window is displayed, the screen area behind it is backed up in EGS program memory. When the Command window is removed (not visible) from the screen, the background area is restored to the previous image present before the Command window became visible.

As with the A and B windows, the Command window's size, shape, and location may be changed by the user at Setup time (WINDOW). The one caution is that the window size must be big enough to hold the largest message (at the current font size) that can appear in the window. **Note:** Not all messages must use the complete window size.

The current default Command window size is 7 text lines (80 characters) and its location is the lower left corner of the screen. It is displayed in the overlay(1) planes. The default font type is Roman8, ascii, 8 bit (bold) - 10 x 20 pixels ("/usr/lib/raster/10x20/lp.b.8U.scf").

2.1.3.3 AlphaNumeric Window

The ALPHA window is a permanent window that is used to display radar tracking data in a text format.

As with the Command window, the AlphaNumeric window uses the Fast Alpha/Font Manager library routines to write text data to the screen. Several font sizes are available (see page A-2, 'Fast Alpha Font Manager Programmer's Manual'). The window size is fixed by both the font size and quantity of text displayed in it. If a smaller window is desired, or different screen hardware is used, the font size may be changed.

The contents of this ALPHA window are further explained in Section 2.1.7.8.

As with the previous windows, the ALPHA window's size, shape, and location can be changed by the user at Setup time (WINDOW). The one caution is that the window size must be big enough to hold the selected FORM (at the current font size) that appears in the window.

Also, at Setup time the ALPHA window's predefined FORM number may be changed, which allows the user to change the format (type) of tracking data that is displayed.

The current default ALPHA window size is 50 text lines (24 characters) and its location is the right side of the screen. It is displayed in the image(0) planes. The default font type is Roman8, ascii, 8 bit (bold) - 10 x 20 pixels ("/usr/lib/raster/10x20/lp.b.8U.scf").

2.1.3.4 AER and REF Windows

Two special purpose (multi-use) windows are defined to display text for certain Control Panel commands. The Azimuth, Elevation, and Range (**AER**) window shows the relative position of the Radar A track to other points (Reference, Mouse). The REference window displays the current reference point coordinates, the X-Y coordinates of the mouse in the 'Line Draw' function, or the Wind values during the 'Wind Run' function.

Both the AER and REF windows are 'pop-up' text windows that are only visible when their associated functions are active. Each window will display only one function at a time.

Fast Alpha/Font Manager library routines are used to display the text on the screen. Several font sizes are available (see page A-2, 'Fast Alpha Font Manager Programmer's Manual').

During the time these AER or REF windows are displayed, the screen area behind each is stored in EGS program memory. When the window is cleared, the background area is restored to the previous image.

Setup mode (WINDOW) can be used to change either window's size, shape, or location. The one caution is that the window size must be big enough to hold the maximum quantity (format) of text required for the window. Not all messages must use the complete window area.

The current AER window default size is 6 text lines (20 characters) and its location is the upper left corner of the screen.

The current REF window default size is 10 text lines (42 characters) and its location is the upper right corner of the A window.

Both the AER and REF windows use the overlay(1) planes by default.

The default font type for both windows is Roman8 and ascii ("/usr/lib/raster/10x20/lp.b.8U.scf").

2.1.4 Input Devices

Two types of user input devices are implemented with the Graphics Real-Time Program (GRTP), the 'keyboard' device and the 'locator' device (mouse). Both of these input devices are opened as Starbase choice devices and respond as Starbase events.

Starbase events use UNIX software signals (SIGUSR1[16] and SIGUSR2[17]). When Starbase events are enabled (sb_deamon_9.0) these signals are not available for use by the GRTP for any other purpose (i.e., pollpio signals). Therefore, the GRTP attached to 'Target' shared memory must be in the 'READ_ONLY' (asynchronous) mode.

2.1.4.1 Keyboard (Function Keys)

The keyboard on the HP370 is opened as a Starbase 'keyboard' device. As such, it retains most of the features of the Internal Terminal Emulator (ITE) for HP terminal (bit-mapped display and HP-HIL keyboard). In other words, the User Function keys and standard escape sequences are defined. Also, the Break (Cntrl-C) key and other terminal functions are implemented. All of the standard "ASCII" character set is active as well as the special function keys (Previous, Next, Home, Arrow keys).

The Graphics Real-Time Program (GRTP) has its own event decoder routine (Check_Events) that parses the input keyboard characters and forms the text strings. Also, certain keys have been defined to perform special tasks as follows:

Key	Character(s)	Function
User Function (f1)	Esc-p	Clear AER window
User Function (f2)	Esc-q	Clear REF window
User Function (f3)	Esc-r	Nothing
User Function (f4)	Esc-s	Wind Run
User Function (f5)	Esc-t	Stop Wind
User Function (f6)	Esc-u	Nothing
User Function (f7)	Esc-v	Nothing
User Function (f8)	Esc-w	Quit the GRTP program

Note: The 'Menu' key turns the User Function key display on or off.

Window Control Keys (Pan / Zoom)

CTRL-A	\01'	Set Pan/Zoom keys to A window (A window must be active)
CTRL-B	\02'	Set Pan/Zoom keys to B window (B window must be active)

Pan/Zoom Keys

Home (↖)	Esc-H	Reset current 'View' window to start size and location
Previous Page	Esc-V	Zoom Out
Next Page	Esc-U	Zoom In
Up Arrow (↑)	Esc-A	Pan Up
Down Arrow (↓)	Esc-B	Pan Down
Right Arrow (→)	Esc-C	Pan Right
Left Arrow (←)	Esc-D	Pan Left

2.1.4.2 Mouse

The mouse on the HP370 is opened as a Starbase 'locator' device. As such, the mouse cursor location can be referenced to the current Starbase World Coordinate (wc) system and the current modeling transformation matrix.

Input from the 'locator' device is enabled as events in the Graphics Real-Time Program (GRTP). When the 'locator' events are enabled, the mouse position is slaved to a cursor in the A window (tracking mode). The mouse buttons(3) are read as choice events by the GRTP routine (Check_Events). When a

button is pressed, the button number (1, 2, 3) and the current position (virtual coordinates) are returned. The virtual coordinates (vc) are converted to world coordinates for calculation purposes.

The 'locator' device events are not normally enabled in the GRTP, but they are enabled for two functions (Mouse Line Draw and [AER] Mouse to Radar A).

2.1.5 Master Process

The 'EGS_master' program is a set of 'C' language functions that start and control the HP370 programs. It is the supervisor process for the Command shared memory, the Command decoder process, and the graphics process (either GRTP or Setup). A block diagram for the 'EGS_master' main program is shown in Appendix L (Figure L.1).

The main portion of the 'EGS_master' program creates the Command shared memory and attaches to it. Next, this Command shared memory block is initiated to known default values by the function 'set_default_values' as shown in Appendix L (Figure L.2).

2.1.5.1 Startup Files

The last set of HP370 Setup parameters that were saved are stored in four disk files. These parameters are used as the new Startup values each time the 'EGS_master' program is started. Function 'Init_Shm_Memory' reads these disk files and stores the values in Command shared memory.

Only one disk filename, "STADEF," is fixed in the entire set of HP370 programs. This **Start Default** (STADEF) file is checked for the other filenames whenever 'EGS_master' is started and each time the Setup (X11) child process stops. Any of these filenames may be changed during Setup mode by the START client.

The "STADEF" name cannot be changed; but another file with the same format can overwrite it. Caution must be exercised when performing this task because the old contents could be lost. A safer way to change Start files is to first save the old file under a different name, then rename the new file to "STADEF".

As previously mentioned, "STADEF" contains the names of the three other disk files. These disk files are the window file, the radar file, and the view file. They are used by the real-time plotting program to define the display windows, radar plot symbols, and view contents. Initially, they were given the names **Window Default** (WINDEF), **Radar Default** (RADDEF), and **View Default** (VEWDEF). There is nothing magical about these names and the current filenames can be changed with the START client. Each of the default files (window, radar, and view) may be edited in the Setup mode by the WINDOW, RADAR, and VIEW clients respectively. A complete discussion for each client function is given in Section 2.2.

2.1.5.2 Child Processes

After initiating Command shared memory, the 'EGS_master' program starts two other processes. First it starts the Command decoding process 'Command_proc' and then it starts the real-time plotting process 'Graphics_proc'.

2.0 Operational Modes

To start both of these child processes, the UNIX 'fork' command is used to spawn a new process and the UNIX 'execvp' command is used to begin executing the appropriate file ('Command_proc' or 'Graphics_proc').

After each child process starts executing, the 'EGS_master' process suspends itself until a child stops or is terminated. This is done with the UNIX 'wait' command. If either of the child processes stops or is terminated the 'EGS_master' is notified (restarted).

If the stopped process is 'Graphics_proc' and the current operation mode is Setup (Setup command received from the Control Panel), another child is created. The script "x11setup" is called to start the Setup display (X11 Server).

When the stopped process is 'Command_proc' or the current operational mode is not Setup (Stop Panel -> QUIT), the master program stops any remaining child processes, cleans up Command shared memory, and stops executing.

2.1.5.3 X11 Setup Process

The "x11setup" script calls the HP X11/Motif Window Server and passes it the names of the client programs that are to be started.

On the HP370, the time required to start the X11/Motif Window Server with clients is fairly long (about 60 seconds). The Server must be allowed to complete initialization or bad things happen (it crashes). Consequently, 'EGS_master' locks out any Control Panel commands for 60 seconds from the time the X11 Server is started. Note, the 'Command_proc' child is still active and commands can be received from the VAX. However, they will not be acted upon until this time-out has occurred.

Once the Setup mode is entered, 'EGS_master' responds differently from the normal Real-time mode. It does not suspend execution, but every 2 seconds it checks the current programs ('Command_proc' and X11 Server) status.

If the Setup process (X11 Server) terminates, the real-time process 'Graphics_proc' is restarted.

When the operational mode changes from Setup (Realtime or Playback) the Setup process is stopped and the real-time process 'Graphics_proc' is restarted.

2.1.6 Command Process

The 'Command_proc' program is a set of 'C' language functions that reads commands from the VAX and decodes them. It sets parameters in the Command shared memory and/or returns values to the VAX. A block diagram for the 'Command_proc' main program is shown in Appendix L (Figure L.4).

The main portion of the 'Command_proc' program attaches to both Command shared memory, Target shared memory, and Reply shared memory. Target/Reply shared memory blocks are described in Appendix B. The Command shared memory block is discussed in Appendix C.

2.1.6.1 Command String

The command string is received from the VAX computer over the parallel interface (gpio). It is read by the 'pollgpio' driver, which places it in the Target shared memory block. In addition, the 'pollgpio'

daemon sends a UNIX signal (USRSIG1) to all processes that are attached in synchronous mode. The notified process has 0.1 seconds to read the new data before it will be overwritten.

In the 'Command_proc' program the function "wait_for_it" is used to suspend execution for 120 milliseconds (maximum) per decoding cycle. If the signal (USRSIG1) is received before the time limit has elapsed, the Target command string is checked at that time. This is the mechanism (USRSIG1) by which the Command process and the received data are synchronized. Also, this assures that a command will never be missed.

When a new Target shared memory block is signaled, the Command process checks to see if a command is present. It looks for the ASCII character STX (Start of Text - 0x02) followed by either an ASCII 'B'(0x42) or an ASCII 'G'(0x47). If either of these two character sequences is found a command string is assumed to follow. Each valid command sequence must be followed by a Null command (0x00) string before a new sequence will be recognized. This ensures that a communication error does not cause the same command to be repeated over and over.

The function "check_cmd" checks the remaining command string and sets the appropriate value in Command shared memory.

Three immediate mode commands (Op. Number, View Names, and Start Panel) do not set values in Command shared memory.

Instead, they return information from Command shared memory to the VAX by Reply shared memory. The Start Panel command (V1) clears the Reply shared memory Write Flag for initialization purposes.

At the present time, a total of 37 separate command functions are decoded by the "check_cmd" function (Appendix L - Figure L.5).

2.1.6.2 Handshaking

When a valid command is decoded in the "check_cmd" function it sets the appropriate parameter in Command shared memory. At the same time it sets the 'command_flag' value to True(1). This indicates to the graphics process that something has changed in Command shared memory. The graphics process checks Command share memory, performs the indicated function, and clears the 'command_flag' value(0).

Two commands require a response from the graphics process. They are **Time Search** request and **Reference Point** coordinates request. These two commands cause the Graphics Real-Time Program (GRTP) to pop up a window on the screen. This window prompts the operator to enter new values from the keyboard. When the operator is finished entering the requested information, the pop-up window is removed and the values are set in Command shared memory. At the same time the 'graphics_flag' is set(1) to indicate that the request is satisfied. The command process can then send the data to the VAX and clear(0) the 'graphics_flag'. Only one request procedure can be active at a time.

Note: The 'Reference Point' request has only been implemented up to returning the string to Command shared memory, but not to sending it to the VAX.

2.1.7 Graphic Display Process (GRTP)

The Graphics Real-Time Program 'Graphics_proc' consists of a set of 'C' language functions that plot symbols whose location is based on the radar tracking data (TVTS). These symbols are overlaid on a TTR range map and are scaled accordingly. Multiple views of the same radar track are possible by plotting in different display windows (A and B). Specialized information is calculated and presented to the operator, in text format, in any of three windows (ALPHA, AER, or REF).

Operator interaction/control is implemented by means of the Command window, the keyboard, and the mouse device.

The Graphics Real-Time Program configures the various windows, and what is displayed in them, based upon current parameters as read from Command shared memory. The Command shared memory block is further explained in Appendix C.

Radar tracking data is read from the VAX computer by means of the Target shared memory block. The 'Graphics_proc' program is attached to the Target shared memory in the asynchronous mode and under certain circumstances a data point may be missed. Appendix B shows the format for the Target shared memory.

The main portion of the 'Graphics_proc' program attaches to both Command and Target shared memories. It initializes Starbase graphics devices, filter parameters, list heads, keyboard events, and program timers. Also, the 'Graphics_proc' main function checks Command shared memory for command flags and calls the appropriate functions when they are set. It calls the library plot routines 'Plot_Radar_Sym' to plot tracking symbols with trails. Finally, it calls the current 'alpha_fill' function to update the ALPHA window. A block diagram for the 'Graphics_proc' main program is shown in Appendix M.

2.1.7.1 Initialization

The initialization parameters for the 'Graphics_proc' program are primarily located in two places. The fixed parameters are stored in an include file 'egslib.h' while user changeable parameters are stored in Command shared memory ('Command.h').

The include file 'egslib.h' contains parameters that are fixed in the 'Graphics_proc' program at compile time. These parameters include the A and B radar symbols (colors), maximum array sizes, keyboard definitions, color table definitions, and many more. Also in this file are the 'C' language structure definitions used throughout the HP TTR EGS graphics system. A complete listing of the 'egslib.h' file is included in Appendix J. These parameters may only be changed by editing this file and remaking the 'Graphics_proc' program.

The Command shared memory (Appendix C) parameters are loaded (defaults) each time the 'Egs_master' program is started. They exist as long as the master program exists. Most Command shared memory values are changed by the 'Command_proc' or the 'x11setup' processes. Primarily, the 'Graphics_proc' reads the Command shared memory parameters and uses them to configure the display screen. Notable exceptions to this rule are the command requested values 'Time Search' (string) and 'New Ref. Point'. In addition, the Graphics Real-Time Program (GRTP) saves common parameters in Command shared memory when switching between Real-time and Setup modes. These save parameters are the Window Control Key, View/Segment values, and Pan/Zoom (matrix) parameters.

One of the first steps performed by the 'Graphics_proc' is to initialize (open) the Starbase graphics devices. It opens each of the graphics windows (A, B, Cmd, ALPHA, AER, REF) in either the image or overlay planes. The setup procedure WINDOW is used to set the parameters that determine how (image/overlay) the window is opened, as well the screen location and size. Its values have been stored in Command shared memory by 'Egs_master'. Once the graphic windows have been opened in either the image or overlay planes, they cannot be changed without first closing and then reopening the graphics device. At this time, the 'keyboard' and 'locator' Starbase devices are opened.

The Invisibility filter parameters are cleared such that all segments will be displayed (turned on).

The AlphaNumeric window (ALPHA) does not use Starbase graphics calls to display text. Instead, it uses Fast Alpha/Font Manager library routines to display text quickly. The exact Fonts are loaded and Font/background colors are set in subroutine 'Set_Window_Parms'. Each display for this ALPHA window has been predefined (FORM#) in the forms library 'frmllib1.c' and may be selected at setup time (WINDOW). Each FORM consists of two parts: the static background form 'alpha_form' and the changing portion 'alpha_fill'. The changing part 'alpha_fill' is updated at a once-per-second rate, while the background 'alpha_form' is only redrawn when the complete screen is refreshed. This speeds up the program execution time.

At 'Graphics_proc' initialization the FORM number is read from Command shared memory. Then the corresponding functions ('alpha_form' and 'alpha_fill') are referenced as 'C' pointers to functions. A complete description of the FORMs and form library is given in Section 2.1.7.8.

Trail arrays are large position (x,y,z,h) arrays that are used to remember the locations of the last X number of radar data points. The number X equals the current trail length. Each Trail array is a circular First-In/First-Out (FIFO) buffer with a revolving pointer to the next trail point. As a new radar point is received it is plotted on the screen while the oldest point is removed. Up to 10 Trail arrays (8 radars + 2 current A and B radars) are maintained by the program. The number of points in each Trail array is determined by the TrailMax value in the 'egslib.h' include file.

At 'Graphics_proc' initialization the Trail array positions are cleared to a point which is offscreen (hopefully). The contents of the Trail arrays are lost when 'Graphics_proc' terminates.

Linked Lists are 'C' structures that have as their last element a pointer (address) to another structure of the same type. Therefore, they can be used to build variable length arrays of items. The end of the list is determined by a NULL address pointer to the next list cell. Memory can be dynamically added or released ('malloc' and 'free') by the program, subject only to available computer memory. The first structure in the list is called the header cell.

The 'Graphics_proc' program uses the Linked List concept to build variable length arrays for the label operation, the hack operation, and the line draw operation. The contents of the Linked Lists are lost when 'Graphics_proc' terminates.

During the initialization process, the 'keyboard' device is enabled and remains enabled. The 'locator' (mouse) device is only enabled when an operation requires it.

The keyboard User Function keys are programmed with the values found in the 'egslib.h' file by the function 'Set_Softkeys'.

2.0 Operational Modes

Blank A and B windows are displayed on the screen by the function 'Draw_Static_Windows'. Also, the current background FORM 'alpha_form' is sent to the ALPHA window.

An array of the most recent five position (x,y,h) values for the current A and B radars is used when smoothed data is required. These position arrays differ from the current A and B Trail arrays because the x, y, and h values are referenced to range zero, while the plotted Trail arrays may not. The smoothing array will always be present while the Trail array may not (trail length = 0).

The update rate for these smoothing arrays is 0.1 seconds per point. The smoothing array is used to calculate a moving average over the five data points and thus minimize the effect of an erroneous data spike. The disadvantage is it may produce smaller errors that take more time to settle out. At this time it is only used in the FORM#3 heading calculations and the 'Wind Run' calculations.

Four timers are running in the 'Graphics_proc' main control loop. They are the peck rate (trail) timer, the loop timer (0.1 second), the AlphaNumeric (1.0 second), and the blink timer. Each of these timers is referenced to the HP370 Unix clock and has a resolution of 20 microseconds. The actual time value egs_time uses the 'C' language function 'clock' and is relative to the start of program execution.

The loop timer (0.1 second) is used to compute a variable delay time. If the last pass through the control loop exceeds 0.1 seconds, a marker (*) is flashed to the screen and execution continues. However, if the last pass through the loop was less than 0.1 seconds a difference value is calculated and the function 'wait_for_it' is called. This function waits the specified amount of time, then continues loop execution. The resolution of this function clock is 20 microseconds.

2.1.7.2 Command Implementation

Graphics Real-Time Program commands are initiated from two sources. The first source is the PC Control Panel and the second source is the local HP370 keyboard (or mouse).

The PC Control Panel external commands are the commands that have been discussed earlier in this document. They are initiated at the PC front panel and are passed through the VAX computer to the HP370 computer. The program 'Command_proc' interprets them and sets a flag in Command shared memory.

Internal commands are initiated at the HP370 computer by the keyboard or mouse (locator) devices. They are implemented as Starbase events and are checked every pass through the control loop.

Many operations on the TTR EGS graphics system are combinations of these two command types. For instance, the **Time Search** command is started by a get start time (A6) at the PC Control Panel. It is completed by the HP370 computer popping up the Cmd window and displaying a prompt message. When the operator finishes typing the time value into the computer, the response is placed in Command shared memory. The 'Command_proc' program detects this completion (graphics_flag) and returns the time string to the VAX computer.

Each key stroke at the HP370 computer generates a Starbase keyboard (choice) event in the 'Graphics_proc' program. The program knows which keyboard process is active (Time Search) and builds the time string. When a carriage return (CR/LF) is detected the string is complete. After the time string has been entered, a verify prompt is displayed. Answering Yes returns the string and clears the current active keyboard request.

This operation is implemented by the function 'Time_Search_Value' (Appendix M - Figure 2.1.7.6) and by the library function 'get_key_command'.

In addition to the external and internal designations, the commands may be further categorized as direct and implied commands. Direct commands cause an action to be taken by the graphics program (a function is called). Implied commands are performed during the process of redrawing the display screen with new parameters (refreshing).

Direct commands are implemented by branch statements in the main control program 'Graphics_proc' or in the event function 'Check_events'.

Implied commands are implemented during the call to the function 'Refresh_Screen'. The 'command_flag' being set (1) is the trigger that causes the screen to be redrawn and implied commands completed.

The **Operate Mode** parameter in Command shared memory is a character string that indicates the current EGS system mode. If the Operate Mode is "Real-time" or "Playback", the HP370 graphics system performs its normal plotting functions. An Operate Mode of "Timesrch" causes the main program to call the 'Time_Search_Value' function and return a time string to Command shared memory. The string "Quit" in the Operate Mode parameter causes the 'Graphics_proc' program to clean up all of its active resources and terminate program execution with a zero(0) status.

If the Operate Mode is "Setup", the 'Graphics_proc' program cleans up resources and terminates with a status of two (2). This termination of a child process wakes up the 'Egs_master' program and the status (2) tells it to start the X11 server 'x11setup' process (Section 2.2).

The **Label flag** (TRUE) causes a marker, the current time, and the current altitude to be displayed at the track location of Radar A in both the A and B windows. This is a linked list and any number of labels may be displayed.

The **Label Clear** flag frees all of the Label linked list memory, except the header cell, and refreshes the display screen. This causes the screen to be redrawn without any labels.

The **Hack flag** (TRUE) causes a marker to be displayed at the position of all selected radars in both the A and B windows. There may be 8 radars (maximum) per window. This is a linked list and any number of hacks may be displayed.

The **Hack Clear** flag frees all of the Hack linked list memory, except the header cell, and refreshes the display screen. This causes the screen to be redrawn without any hacks.

Line Draw mode is initiated by setting the Line parameter to either one (1-mouse) or two (2-keyboard) for the input method. Function 'Draw_Line' is called to start the line drawing process. Lines can only be drawn in the A window. Line coordinates are kept in a linked list array and any number of vertices can be saved in the array.

During line draw mode, all the normal graphics plotting operations are continued in real time. However, the displayed view/segments cannot be changed and the screen cannot be refreshed (keyboard only). The line drawing process is further explained in Section 2.1.7.7.

2.0 Operational Modes

Line segments are removed from the screen by clearing(0) the Line parameter. This frees the linked list memory, except the header cell, and refreshes the display.

The **Hardcopy** flag (TRUE) causes the current A window bitmap to be saved in a file. Then the line printer spooler is invoked to plot it on the graphics printer. Even with spooling this is a fairly lengthy (30 seconds) process and the display screen is locked during this time.

Azimuth, Elevation, and Range (AER) operation is initiated by setting the AER to X parameter to one of the following:

- 1 → Radar A to Reference Point
- 2 → Reference Point to Radar A
- 3 → Current Mouse location to Radar A

Function 'Display_AER' is called to start the AER window display process. All normal graphics plotting operations are continued in real time. Once it is initialized, the AER window is updated at a 1.0 second rate (AlphaNumeric timer). The AER process is further explained in Section 2.1.7.6.

AER operation is halted by pressing the User Function key (F1) on the HP370 keyboard.

The **Show Reference Point** flag (TRUE) causes the currently selected reference point to be displayed in the REF window (popup). The REF window is used by the Show Reference Point command, the Line Draw (mouse) operation, and the Wind Run operation. These are mutually exclusive operations and are further explained in Sections 2.1.7.7 and 2.1.7.9.

REF window operation is terminated by pressing User Function key (F2) on the HP370 keyboard.

The **Reference Point Modify** command is initiated by setting the Ref. Point parameter equal to nine (9). Function 'Get_RefIn' is called to start the Ref. Point keyboard operation. This function pops up the Cmd window and displays a prompt message. All the normal graphics plotting operations are continued in real time. However, the screen cannot be refreshed during the time the Cmd window is visible.

When the operator finishes typing the new reference point string, it is parsed and placed in the Command shared memory structure reference. The 'Command_proc' can detect this completion by monitoring the graphics flag.

Each key stroke on the HP370 keyboard generates a Starbase keyboard (choice) event in the 'Graphics_proc' program. The program knows that the Reference Point process is active and builds the reference string. When a carriage return (CR/LF) is detected the string is complete. After the reference string has been entered, a verify prompt is displayed. Answering Yes returns the parsed string and clears the current active keyboard request.

The Single Point Coordinate flags (TRUE), IN1/OUT1 and IN2/OUT2, cause a marker to be drawn at the specified coordinate location. The marker is drawn in both the A and B windows.

The Single Point Coordinates are removed by clearing (FALSE) the coordinate flags IN1/OUT1 and IN2/OUT2.

2.1.7.3 Screen Refresh

As mentioned previously, implied commands are implemented by redrawing the display screen (A and B windows) with the newest parameters. The 'command_flag' being set(1) in Command shared memory causes the function 'Refresh_Screen' to be called.

The 'Refresh_Screen' function completely clears the A and B windows. Then it draws each window based on the current setup parameters from Command shared memory.

First the current view number for each window (A and B) is checked. If the view number has changed, the current Starbase/Display List image is deleted from memory. The new Display List view is loaded from a disk file (*.vfl) in segment integer format. Next the segment names and namesets are edited to make them agree with the hierarchical form shown in Appendix D. Editing the filter parameters and segment calls at this point allows the same view file (*.vfl) to be used in multiple views in the real-time program (GRTP).

Also, the current translation matrix for the view number is pushed onto the Starbase matrix when a new view is loaded. If this is the first time this view has been selected, the matrix represents the initial values that were set in the VIEW setup procedure. Otherwise the translation matrix represents the state (pan and zoom) of the view the last time it was visible (selected).

The current window (A or B) view is sent to the display screen by traversing the Display List in memory. This traversal is very fast once the Starbase image is in memory. While the Display List is being traversed each segment (nameset) is compared with the current Invisibility Filter bits. This determines which of the segments(8) are on or off. The Invisibility Filter bits are set/cleared depending on the current values in Command shared memory.

For the A and B windows the current window size (X and Y) is calculated in modeling coordinates (meters) and is saved for use by other parts of the graphics program. The A window size is converted to the current user units (feet, meters, or nautical miles) and is displayed on the screen (lower, right).

The trail arrays used in the Graphics Real-Time Program (GRTP) are First-In, First-Out (FIFO) circular buffer arrays. Instead of the complete buffer data values shifting each time a point is updated, a revolving location pointer is used to address the **next plot point** in the buffer. This turns out to also be the **last trail point** to be erased in the circular array. As the trail length is expanded (longer) or contracted (shorter) the length of the circular buffer is adjusted. This concept works well as single locations are plotted in the loop and the loop length does not change.

Also, a separate array for each active radar(5) and separate arrays for both the A and B radars are maintained by the graphics program. This assures that a trail path is not lost when the current Radar A (or B) is switched in the middle of a test. However, it does lead to a large amount of trail swapping if the A or B Radars are changed often.

Another problem occurs when a Radar is **not** active or is turned off for awhile. When it is turned back on, the trail array for that Radar may contain stale (old) position data. To avoid the stale location data problem, the trail array for that Radar is cleared whenever it is **not** active.

The biggest problem with the circular FIFO trail array technique occurs when the trail length changes size (longer or shorter). When the trail length gets longer a piece of blank array must be inserted into the loop corresponding to the increased trail size. Likewise, when the trail length is shortened a portion of

the circular buffer must be removed. Luckily, the site for these additions (or deletions) is at the **next plot point** location pointer. This does require shifting of the array values and pointers, but hopefully not very often.

After the trail arrays have been configured, the symbol and trail markers for each Radar (A, B, all) are plotted. This plotting is done in both the A and B windows. Trail marker updates are performed with the Starbase 'polymarker2d' function call. This routine is very fast when drawing a large number (fixed array size) of marker symbols.

Next, any Labels, Hacks, and Line segments are redrawn on the screen. These are Linked Lists with a variable number of data locations. To draw each Label (Hack or Line) the List is traversed one element at a time. If the Lists are very long, this can significantly slow the refresh execution rate.

Finally, Input Symbols (#1 or #2) are re-displayed if needed.

2.1.7.4 Starbase Events

Internal commands are initiated at the HP370 computer by the keyboard or mouse (locator) devices. They are implemented as Starbase events and are checked every pass through the control loop. The Graphics Real-Time function that performs this event check and initiates the desired action is called 'Check_Events'.

When enabled, Starbase events are triggered by the operator pressing a key on the keyboard or a button on the mouse. Each keypress generates an interrupt to the Starbase daemon program 'sb_daemon_9.0', which saves it in an **event queue**. In fact, Starbase uses Unix signals (SIGUSR1[16] and SIGUSR2[17]) for this purpose. Each pass through the control loop the Graphics Real-Time Program (GRTP) checks this queue.

On the HP370 ITE terminal emulator, a key press generates either a one- or a two-character ASCII sequence. For the two-character sequence the first character is always the Escape(033) character and it is used to initiate internal commands on the HP370. These special keys consist of the User Function (F1 - F8) keys, the Arrow keys, the Home key, the Previous Page key, and the Next Page key.

The control key used in conjunction with either the A or B keys is used to change the window (A or B) to which the Pan and Zoom key actions are directed.

User Function keys (F1 - F8) are programmed by the Graphics Real-Time Program (GRTP) to initiate or complete specific tasks. The F1 key is used to terminate the Azimuth, Elevation, and Range (AER) window functions. Key F2 removes the REference window from the screen. The F4 key initiates a 'Wind Run', while the F5 key terminates a 'Wind Run'. The F8 key initiates a program halt sequence (Quit) if properly verified.

The Pan and Zoom key set consists of the Arrow keys, the Home key, the Previous Page key, and the Next Page key. When one of these keys is pressed, they cause the Starbase stack matrix to be altered by a fixed percentage of the current window size. This gives the appearance of moving the current window's (A or B) viewport by the same fixed amount. The arrow keys cause a lateral or vertical translation in the direction of the arrows. The Previous key causes the image to become smaller (zoom out), while the Next key causes the image to become larger (zoom in). The Home key resets the translation matrix to the initial values found in the setup procedure VIEW. The Pan and Zoom keyboard set also force a screen refresh cycle.

If only a single character is detected in the keyboard event, the function checks for any active keyboard process. At the present time only two active keyboard processes are implemented: Keyboard Line Draw and new Reference Point string. In both cases the characters are stored in a string until a Carriage Return <CR> is detected. Then the string is parsed and the appropriate action is taken (draws the Line Segment). The advantage to this technique is that real-time plotting is not disturbed while keyboard input takes place.

Mouse (locator) button events occur when the operator presses one of the mouse buttons (1, 2, or 3). While keyboard events are always enabled, the locator events are only enabled in response to an external command procedure (Mouse Draw Line or Mouse AER window). In general, mouse button #2 is used to halt the Line drawing input procedure but not erase the lines. Button #1 specifies to draw a line segment from the previous location to the current position of the mouse cursor. Mouse button #3 specifies a move from the previous location to the current position of the mouse cursor but does not add a line segment. Pressing any mouse button in the Mouse AER window procedure causes an immediate window update to occur, as compared with the normal 1-second update rate.

2.1.7.5 Radar Symbol Plotting

All plotting of the Radar tracks is performed by calling the the library function 'Plot_Radar_Sym'. It is an extremely versatile function that plots any of the Radars (A, B, or all) in both the A and B windows. It updates the appropriate Trail arrays, utilizes the current setup parameters, and implements the Freeze Screen command. It reads the selected Radar position directly from the Target shared memory block.

The function 'Plot_Radar_Sym' uses the default values defined in the include file 'egslib.h' to select the Radar A (and Radar B) marker type, marker size, and marker color. The Trail size is always selected to be one quarter (.25) of the symbol size. Any of the other Radar parameters, except for Radars A and B, are read from command shared memory and may be changed by the RADAR procedure in the setup mode of operation.

Actual A and B window axis position data (X and Y) is selected by this function. In the setup procedure WINDOW, the operator has a choice as to what type of Radar data can be plotted on the X-axis and the Y-axis. This function implements those selections by reading the 'Command' shared memory 'xplot' and 'yplot' parameters.

This 'Plot_Radar_Sym' function takes care of all the overhead involved in erasing the old Radar plot symbol and the oldest Trail marker. Then it draws a new Trail marker at the last plot location. Finally it draws a new Radar symbol at the current Radar position.

If either the A or B window has been turned off (not active), no window updating is done. Also, the portion of the Trail arrays reserved for that window is **not** updated.

When the **Freeze Screen** command is in effect the screen writes are skipped. However, the Trail array updates are continued and when the command is removed the graphics program resumes plotting in the normal way.

2.1.7.6 AER Function

The Azimuth, Elevation, and Range (AER) function is an external command that is initiated from the PC Control Panel. It can be started in any one of three display modes. No switching between modes is allowed once the AER function is operating.

2.0 Operational Modes

1. Radar A to Ref. Point - the initial point of the AER vector is the current Radar A track point, while the terminal point is the currently selected reference point.
2. Ref. Point to Radar A - the initial point of the AER vector is the currently selected reference point, while the terminal point is the current Radar A track point.
3. Mouse to Radar A - the initial point of the AER vector is the currently displayed Mouse cursor location, while the terminal point is the current Radar A track point.

In all three cases a pop-up window (AER) is displayed on the view screen and the selected alphanumeric vector information is displayed in the window. Vector data is displayed at a once-per-second update rate as long as the AER window is active. The Mouse mode allows an update whenever a mouse button (1, 2, or 3) is pressed as well as the 1-second rate.

The AER window is removed by pressing the F1 User Function key.

Displayed units for the Range value are the currently selected user units (feet, meters, or nautical miles). Units for Azimuth (0 - north) and Elevation (0 - horizontal) are always in degrees.

The complete AER operation is implemented through three EGS library (egslib.a) functions:

- 'Display_AER' - sets up the AER window and maintains bookkeeping control values.
- 'xyz_rae' - calculates the Azimuth, Elevation, and Range between two TTR range coordinate (x,y,z) points.
- 'Clear_AER' - removes the AER window, restores the background, and clears control flags.

2.1.7.7 REF and Line Draw Functions

The REFerence window is used for three purposes in the EGS Graphics Real-Time Program (GRTP). It is used to display the current Reference Point coordinates, to display the Mouse cursor coordinates during the Line Draw function, and to display Wind zone information (Section 2.1.7.9).

During the **Show Ref. Point** command the function 'Display_REF' pops up a window (REF) on the view screen and displays the ref. point TTR range coordinates (x, y, z, h) in meters. The alphanumeric text is displayed once and is **not** updated. The pop-up window (REF) remains on the screen until it is removed by pressing the F2 User Function key.

The **Line Draw** command permits the operator to draw line segments on the graphics screen (A window) in real time. The lines are maintained between changes in views as long as the Graphics Real-Time Program (GRTP) is executing. They are lost when the graphics procedure terminates.

The Line Draw function can be started in either of two input modes (or both simultaneously):

1. Mouse Input - line segment endpoints (vertices) are entered with the Mouse cursor identifying the location and the Mouse button #1 (draw) or button #3 (move) signifying ready.
2. Keyboard Input - line segment endpoints (vertices) are entered via the keyboard in the CMD window.

The first coordinate pair <X,Y> entered is always a **move** to that point. Subsequent entries may be either **moves** or **draws**; the exact meaning is as follows:

	Move	Draw	Stop Input
Mouse Button	#3	#1	#2
Keyboard M/D flag	'M' or 'm'	'D' or 'd' <space>	Blank Line <CR>

Note: The **Stop Input** entry only halts any further vertices entries; it does not erase any existing line segments. Only a **Line Clear** command will clear all line segments from the graphics screen.

In order to allow line vertex entry from both Mouse and Keyboard Input simultaneously, both Line Input commands (Mouse and Keyboard) must be received from the PC Control Panel before the first **move** coordinate is entered. Once the first point has been entered, input from any other unselected sources **cannot** be initiated.

The Mouse Line Draw mode displays the current mouse cursor in the A window. At the same time, it will display the current cursor location as numeric text in the pop-up REF window. The display units for the mouse cursor are the current user units (feet, meters, or nautical miles). If the REF window is used by another function (Show Ref. Point) the display portion will be skipped. However, the rest of the Mouse Line Input function will still work. Mouse button #2 stops any further mouse input, removes the cursor, and removes the pop-up REF window.

The Keyboard Line Draw mode pops up the CMD window and prompts the operator to enter <X,Y> coordinate pairs. If the CMD window is used by another function (Time Search or Quit) this Keyboard Line Draw command is ignored until the window becomes available. The input units are the current user units (feet, meters, or nautical miles).

All keyboard characters are stored in a string until a complete line has been entered. Then the line is parsed for an <X,Y> coordinate pair and a Move/Draw flag. Depending on the input string a line segment may either be drawn or a move executed. Either way the line endpoint coordinate (with M/D flag) is stored in the Line Linked List array. A blank line (just a <CR>) stops any further Line keyboard input and removes the popup CMD window.

When both Mouse and Keyboard Input modes are active simultaneously, the endpoint coordinate pairs <X,Y> can be intermixed without any problems. The first point entered from either source becomes the **move** coordinate.

2.1.7.8 AlphaNumeric Data Display

The ALPHA window is a permanent window that is used to display radar data in text format. It uses the Fast Alpha/Font Manager library routines to write text data to the screen.

At Setup time (WINDOW) the user may change the ALPHA window's size, shape, and location on the display screen. It is the responsibility of the user to ensure that the window size is big enough to hold the current FORM contents. The default ALPHA window will hold 50 lines of text with 24 characters per line. One of five FORMs may be selected during Setup.

Each display for the ALPHA window has a predefined FORM number in the forms library 'frmlib1.c'. These FORMs are 'C' subroutines that utilize Font Manager calls to build the ALPHA window display. In the Graphics Real-Time Program (GRTP), the ALPHA functions are referenced as 'C' pointers to functions. This allows a different function to be called from the library depending on the current FORM# value in Command shared memory.

In the forms library 'frmlib1.c' each FORM consists of two parts. The first function (form#) builds the window background while the second function (form#_fill) displays the dynamically changing radar data.

The first function (form#) draws the static background portion of the ALPHA window. It displays the non-changing text and the background color. Also, this function (form#) saves the write locations for each data value used in the subsequent calls to the second function (form#_fill). Consequently, this first function must always be called (at least once) before the second function is called. Once the 'form#' function has been called the 'form#_fill' function may be called any number of times. After the initial call the first function (form#) need only be called when a complete screen draw (refresh) is required. The write locations are always stored in device (screen) coordinates.

The second function (form#_fill) displays the changing radar data in text format. It only overwrites the portion of the display where the changing data resides. This process is much faster than redrawing the complete ALPHA window whenever new radar data arrives. The 'form#_fill' function references both Target and Command shared memory to display the latest data values in the currently selected user units (feet, meters, or nautical miles). This function reads the desired values (or calculates the desired result), checks for validity, and formats them into a text string. Then it sends the text to the display window (ALPHA) by using Font Manager subroutine calls.

At the present time, only two FORMs have been developed for the HP EGS graphics system. FORM#3 ('form3' and 'form3_fill') displays radar data, calculated data, and other text that is commonly used for TTR tests. FORM#4 ('form4' and 'form4_fill') is a blank form that fills the ALPHA window when nothing is desired (to be displayed).

The other FORM numbers (1, 2, and 5) are reserved for future expansion purposes. This expansion may be accomplished by writing the appropriate functions ('form#' and 'form#_fill') and updating the corresponding pointer reference calls in the Graphics Real-Time Program (GRTP).

An example (FORM#3) of the two ALPHA window functions ('form3' and 'form3_fill') is given in Appendix M (Figures M.11 and M.12).

2.1.7.9 Wind Run Function

As mentioned in Section 2.1.7.7, the REFERENCE window is used for three purposes in the EGS Graphics Real-Time Program (G RTP). It is used to display the current Reference Point coordinates, to display the Mouse cursor coordinates during the Line Draw function, and to display Wind zone information. Each of these functions are mutually exclusive; only one function can use the REF window at a time. The first two, Ref. Point coordinates and Mouse Line Draw, are described in Section 2.1.7.7.

The **Wind Run** function allows the operator to calculate/display the Wind velocity and direction during selected altitude (height - msl) zones.

Normally, a helium-filled balloon is launched and is tracked by a radar for display purposes. On the EGS graphics display system, this balloon tracking radar must be declared as Radar A for calculation purposes. The Wind data, average velocity and direction, is calculated. Then it is displayed on the CRT screen and the line printer in real time. At the same time all other calculated functions (ALPHA window) and displays (windows A and B) are also active. At the current time, no Wind data is archived to disk files.

The size and location of the REFERENCE window may be changed in the **WINDOW** setup client (Setup Mode). It may be adjusted to display from 0 text lines up to a maximum of 10 Wind data lines. Each line is assumed to be .02 (virtual coordinates) high and .44 (44 characters * .01) wide for the default font size.

At any time the REFERENCE window can be turned off by depressing the 'Clear REF' function key (F2). However, once the REF window is cleared it cannot be re-activated without restarting the Wind Run function.

Likewise, line printer (hp3630a) output may be stopped by answering 'No' to the prompt 'Is the Line Printer Ready?' when the Wind Run Function is initiated. Remember, either the REF window and/or the line printer output must be active or no useful output can be obtained from the 'Wind Run' function. No disk file archive is active at this time.

The Balloon Tracking procedure is initiated by depressing the '**Wind Run**' function key (F4). Note the HP370's function menu display may be turned on/off by pressing the '**Menu**' key on the HP370 keyboard. The first time the F4 key is pressed the line printer and REFERENCE window are opened and enabled. This may be done at any time before the launch of the balloon and requires several seconds or more to initialize everything. The operator must answer the 'Is the Line Printer Ready?' prompt.

Once the '**Wind Run**' function has been initialized, the next press of the 'Wind Run' function key (F4) is assumed to coincide with the launch of the balloon. It starts the timer (t0, x0, y0, h0) for the first measurement zone.

After the balloon launch (t0), each successive press of the 'Wind Run' function key (F4) delineates a new zone (tn, xn, yn, hn) for purposes of calculating average velocity and direction. For a linearly ascending balloon, each zone represents a wind velocity and direction verses altitude (height - msl). Note an internal timer (1 second) limits the minimum time between each successive press of the F4 function key and the minimum zone time.

2.0 Operational Modes

The '**Stop Wind**' function key (F5) calculates the last zone velocity/direction and outputs the values to the printer. It issues a form feed and closes the line printer device. Also, it closes/removes the REFerence window display.

The algorithm that computes the wind velocity and direction uses a moving average of the most recent five (5) position values (x, y, h) when finding the zone end points. This minimizes the effect of a spurious position data value.

Note the currently designated Radar A may be changed in the middle of a 'Wind Run' without any adverse consequences. However, both Radars should be tracking the same balloon or the resulting wind velocity and direction values are probably meaningless.

2.2 Setup Mode

2.2.1 Purpose

The HP Setup mode allows an operator to configure many of the Graphics Real-Time Program ('Graphics_proc') display parameters. These parameters include the window (A, B, ALPHA, etc.) size and locations. Also the Radar plot symbol parameters and the A and B window View (Switch) segments may be selected.

In addition, the Setup mode is used to create (edit) Segment or View files.

Note: The Setup mode and the Graphics Real-Time plotting mode are mutually exclusive programs. They cannot use the display screen simultaneously.

2.2.2 Required Processes

As in the Real-time mode, four software processes (programs) must be executing in the HP370 Graphics system.

The 'Egs_master' process starts the child processes and coordinates the active processes. It starts the 'x11setup' process when the 'Graphics_proc' dies with a status of 0x200. Also, it kills the 'x11setup' process when an operate mode not equal to Setup is detected. After the X11 Server shuts down, it may restart the Graphics Real-Time Process (Real-time or Playback mode) or simply stop everything (Quit mode).

The 'Command_proc' process and the 'pollgpio' daemon continue functioning just as they did when the Real-time display was active. This keeps open the I/O channel to the MicroVax and the PC Control Panel. In fact, the command string is still decoded and Command shared memory parameters may be set. However, they are usually not executed. The one exception is the Operate Mode parameter that is monitored by the 'Egs_master' program and each of the X11 Server clients. This allows each client to shut itself down and orderly clean up any memory references.

The 'x11setup' script starts the X11 Server/OSF Window Manager and passes it the names of several client programs. Each of these clients is an independent program that operates in a separate X Window. Figure 2.2.2 is a block diagram of all the Server clients and the function(s) they implement.

Each of the previously mentioned processes communicates with the other processes by means of a Unix operation called shared memory blocks. This operation is further documented under the 'inter-process communication (ipc)' heading in the Unix literature. These blocks are fixed memory locations (sizes) that are accessible by several different programs.

The exact structure for the 'Target/Reply' shared memory block is described in Appendix B.

Appendix C details the 'Command' shared memory block structure.

Figure 2.2.1. Setup Software Processes and Intercommunication

Figure 2.2.2. X11 Setup Mode Block Diagram

2.2.3 X11 Windows (Clients)

2.2.3.1 Standard X11 Window Operations

The Setup mode users interface is implemented using the HP X11/OSF Motif Window Manager. A number of X Window clients are started when the X11 Server is initialized. Each client is an independent window-smart program that is designed to perform a specific task. A good tutorial on X11 Window operations is the "A Beginner's Guide to the X Window System" (HP Part# 98594-90002) book.

All normal X Window operations may be applied to each client. This includes sizing, moving, restoring, minimizing, closing, and many more. When the X11 Server is first started all of the clients, except START, appear as icons at the bottom of the display. They may be restored (make screen menu visible) by double-clicking the mouse button on the appropriate icon.

The current active X11 window (client) is the window with the white border around it (as contrasted with gray). This means that any keyboard input will be directed to this client. To change the active window move the mouse pointer to the desired window (icon) and click a mouse button.

At any time a new user 'login' window may be created by clicking the New Window entry in the Root Menu. This is a full hp terminal (hpterm) window that allows the user all the Unix command capabilities just as if they logged in on a separate terminal. The person logging in must have a valid user account and password.

Another client program is called 'xhpcal' and emulates the functions available on an HP41 calculator. This client may be used for general purpose calculations.

2.2.3.2 Sandia Client Format

All of the in-house Sandia designed client programs follow a general format as shown in Figure 2.2.3. They are constructed using X11 library functions (Xlib) and Xt Widgets and Intrinsics functions.

The general format for each client display is a specialized Menu area at the top and middle of the X Window. This area is populated with menu boxes that are to be filled with text information required to complete each client specific task. These menu boxes consist of two types: Edit Boxes and Select Boxes. The user may move between various boxes by using the Mouse pointer or using the Arrow keys to designate the desired box.

Edit Boxes have a dark blue background and are filled with user typed (keyboard) text. The actual information requested is indicated by the label at the left side of each box. The **backspace** key is used to register the text cursor in the selected box and to remove any existing text. Text (numbers) must be entered as a complete line because character mode edits (insert or delete) do not function within the Menu Boxes. The actual text of an Edit Box is not loaded into client program variable until the Edit Box is exited by the Return key, Arrow keys, or the mouse pointer. Therefore, an Edit Box must be explicitly exited to make any change permanent.

Select Boxes have a white background and are changed by pressing the **Tab** key. A valid set of responses will be scrolled through the Select Box as the Tab key is pressed. When the desired answer is displayed in the box, the window must be exited to make it permanent.

Below the specialized menu box area, along the window bottom, is a row of Push Button Boxes. The Push Button Boxes are activated by moving the mouse pointer into the box area and clicking a mouse button. The selected push button will briefly flash white when it has been activated. Each push button performs a specific task in the client program.

The first three push buttons, on the left side of the row, are screen Edit buttons. Push buttons New, Load, and Save perform menu-wide functions and have a dark blue background. The New button clears and/or sets default values in all Menu Edit/Select Boxes. The Load button loads all the menu boxes with values from a disk file, while the Save button stores the current menu box entries to a file. The filename may be entered, by the user, via a pop-up window or a default name (scratch) will be used.

To the right of the Save push button may be client Specific screen Edit buttons. If they are present, these push button(s) have a yellow (orange) background. In general, these client push buttons load certain menu boxes with current values based on the contents of another menu box entry or a client program action. These may be used to fill the menu with current (default) values or to update the screen based on a client program operation.

After the client specific push buttons (right) in the row is the Print push button (green background color). The print function lists the current contents of the screen menu to the line printer. It is mainly used for report or documentation purposes.

To the right of the Print button may be a client specific Execute push button(s). If present, these execute button(s) have a black background. All of the previously mentioned Push Buttons (New thru Print) perform screen (menu) operations and are used to enter/report setup conditions. The client operation (if any) is not actually performed until its Execute button(s) is pressed. Some clients may have none or multiple operations.

Figure 2.2.3. Sandia X Window Client Format

2.0 Operational Modes

After (right) the Execute push button(s) is the Quit push button, which is a red background. The Quit button halts the client program execution, cleans up memory references, and removes the process from the process list. It is **permanent** in this session with the X Server, but will be re-started when the X Server is re-started. Before the client program is actually stopped, a verification (Do you really want to Stop) is requested. Also, if the latest changes to the screen menu have not been saved, a Save Window prompt will be displayed. As a general rule, if the user is unsure whether the client may still be needed, minimize (icon) the client window so that it can be recalled.

The right-most button in the row is the Help Me! push button, which has a gray background. The display help function pops up a shell window that is used to display help information. The green More button displays additional text pages or beeps if there is not any more help. The red Cancel button exits the help screen.

2.2.4 Test Project Setup (START)

The START X Window client works in conjunction with three other client programs: WINDOW, RADAR, and VIEW. Together these four clients configure the Real-time Display screen for a test. They allow the user to easily change window parameters, radar parameters, and views as required. The 'glue' that ties each of these client outputs together and informs the Real-time program of the current setup parameter locations is contained in the START Project client.

2.2.4.1 Test Information

The first four menu blocks in the START menu window (Figure 2.2.4.1) contain test-specific information. Test information is entered into the form and it is stored in the Command shared memory block. This test data is primarily used for documentation purposes in the HP370. However, the **operation number** is returned to the MicroVax in response to the 'Op Number' request (Q1).

The **Test ID** Edit block can contain any combination of up to 30 ASCII characters.

The **Operation no.** block may consist of a maximum of 12 ASCII characters.

The **Test date** menu block has room for 8 ASCII characters in a free-field format.

The **Project Name** block can be up to 14 ASCII characters long and is used as the sub-directory (Project) name. When the Project is 'saved' this directory is created and when the Project is 'removed' it is deleted. This sub-directory is created under the default path '/users/egs/data/Projectname'.

2.2.4.2 Startup File ("STADEF")

As mentioned in Section 2.1.5.1, the 'Egs_master' program looks for a disk file with the name "STADEF" for its setup parameters. This **Start Default** (STADEF) file is checked for other filenames whenever the 'Egs_master' is started and each time the Setup (X11) mode process is exited. This file contains filename pointers to three other disk files as indicated in Figure 2.2.4.2.

Test ID:	Test Id
Operation no:	R123456
Test date:	03/22/94
Project Name:	ProjectTest
WINDOW File:	WINDEF
RADAR File:	RADDEF
VIEW File:	VEWDEF
<input type="button" value="New"/> <input type="button" value="Load"/> <input type="button" value="Save"/> <input type="button" value="Print"/> <input type="button" value="Project"/> <input type="button" value="Quit"/> <input type="button" value="Help me!"/>	

Enter the Project Directory Name:	
Select Project Task:	
<input type="button" value="ProjectTest"/>	<input type="button" value="Save"/> <input type="button" value="Restore"/> <input type="button" value="Remove"/> <input type="button" value="Cancel"/>

Figure 2.2.4.1. START Project Client Format

```
cd /users/egs/data
lp STADEF
```

Test ID	<div style="border: 1px solid black; width: 30px; height: 30px; display: inline-block;"></div>	Test Info. (Command Shared Memory)
R123456		
03/22/94		
ProjectTest	Project Name (Sub-Directory)	
WINDEF	Window File Name	
RADDEF	Radar File Name	
VEWDEF	View File Name	

Figure 2.2.4.2. START (Startup) Files

The "STADEF" name cannot be changed, but another file with the same format can replace it. Caution must be exercised when performing this task because the old contents could be lost. A safer way to change Start files is to first save the old file under a different name, then rename the new file to "STADEF". This file saving and renaming can be accomplished very easily with the Load and Save push buttons at the bottom of the window.

The New push button clears the Test ID, Operation no. Test date, and Project Name Edit blocks. It also installs the default filenames WINDEF, RADDEF, and VEWDEF in the file Edit boxes.

As previously mentioned, "STADEF" contains the names of the three other disk files. These disk files are the window file, the radar file, and the view file. They are used by the real-time plotting program to define the display window, radar plot symbols, and the view contents. Initially, they were given the names **Window Default** (WINDEF), **Radar Default** (RADDEF), and **View Default** (VEWDEF). There is nothing magical about these names and the current filenames can be changed in this START client. This is done by simply entering a new name in the WINDOW file, RADAR file, or VIEW file Edit blocks respectively. The filenames are limited to 9 characters and all characters are forced to upper case ASCII. In addition, the name must be a valid Unix filename whose existence is checked when the Edit box is exited.

Each of the files (window, radar, and view) may be edited in the Setup mode by the WINDOW, RADAR, and VIEW clients respectively.

In order for any changes to take effect in the Real-time program, the menu information must be explicitly saved in the file named "STADEF". This may be done with the Save push button or at the Quit save file prompt.

When the START client is first initialized the current contents of the "STADEF" file are automatically loaded into the menu blocks.

2.2.4.3 Project Menu

The **Project** Push button is a client specific execute push button. Selecting this button causes a second pop-up menu to appear, which is shown at the bottom of Figure 2.2.4.1. This Project menu displays the current Project Name (Sub-Directory), which can be changed at this time by entering a new ASCII Name.

In addition, four functional push buttons are given:

- 'Save' - Copy all the current Test setup files to the Project Sub-Directory.
- 'Restore' - Copy the Project Sub-Directory to the current Test setup files.
- 'Remove' - Remove the Project Sub-Directory and all its files.
- 'Cancel' - Return to the START Main Menu.

The **Save** button first makes the contents of the START menu the current Test setup (STADEF), if it had not been saved. Then it uses the **Start Default** file as an index to the remaining setup files for copying purposes. Note the current Test Setup is not removed during this copy process. If the 'Project Name' subdirectory does not exist, it is created along the default '/users/egs/data' path.

2.0 Operational Modes

All Unique Test setup files are copied during this 'Save' Project function. This includes the Start, Window, Radar, and View default files. In addition all 'View'(*.vfl) and 'Segment' (*.sfl) files are also copied if they are referenced in the View default (VEWDEF) file. The process is completed by following the trail of 'Segment' files that were used to construct the 'View' files. Note all references (*.com, *.asc, *.sfl, and *.vfl) to the 'View/Segment' files are saved to the Project Sub-Directory. This allows the complete reconstruction/editing of the Project at a future time.

One exception to the idea of copying all relevant Setup files is that certain 'Segment' files are used universally by multiple Test Projects. These 'Segment/View' files are declared in an ASCII text file called 'globef.doc'. Any file that has its root name declared in this text file 'globef.doc' will not be saved to the Project sub-directory nor will it be restored from the Project sub-directory. Also, these root names will never be deleted from the main data directory '/users/egs/data/' by these Project functions. Note only the root name needs to be entered in this file because all extensions (*.com, *.asc, *.sfl, and *.vfl) are automatically ignored.

The **Restore** push button first checks to see if the old Test Setup has been saved in a Project sub-directory. If it was not saved it prompts the user to save it. Then this Restore function removes all of the old Project files (except 'globef.doc' files) from the current data directory '/users/egs/data/'. Finally it copies the new setup files from the Project sub-directory to the current data directory. During this function the other X11 setup client programs (WINDOW, RADAR, VIEW) are notified that a new Test Setup has been loaded and they should perform any initialization procedures.

The Setup files with the names 'STADEF', 'WINDEF', 'RADDEF', and 'VEWDEF' are never removed from the current data directory '/users/egs/data/' by these functions. However, they may be overwritten by the Restore function.

Push Button **Remove** deletes all files in the Project sub-directory and then removes the sub-directory itself. Note this function does not remove any files from the current data directory '/users/egs/data/'. To delete the files from this current data directory a new Project must have been reloaded (Restored).

The **Cancel** button removes the Project pop-up window and returns control to the START main menu.

Placing the Test Setup files in a separate Project sub-directory hierarchy facilitates the transferring of Project information between EGS graphics systems. Figure 2.2.4.3 shows the Project directory hierarchy. Appendix N describes script procedures for backing up, restoring, and transferring Project Files between EGS graphics systems over a LAN connection.

2.2.5 Window Configuration (WINDOW)

The WINDOW client allows the operator to set or change the Graphics Real-Time Program (GRTP) screen display parameters. These parameters are the window type, size, location, hardware display planes, plot axis values, and AlphaNumeric display contents. This allows the operator to extensively change the display screen to meet test requirements.

Figure 2.2.4.3. EGS Project Directory Hierarchy

2.2.5.1 Window Coordinates

The HP370 computer system uses an HP98550A graphics display driver card. This driver has a display resolution of 1024 pixels (10.24 inches) vertical by 1280 pixels (12.8 inches) horizontal. As depicted in Figure 2.2.5.1, these are the **device coordinates** for the EGS Graphics System.

In addition the graphics display memory is organized into two layers which are called **image planes** and **overlay planes**. Drawings that are constructed in one set of planes do not physically interfere with drawings constructed in the other set of planes.

The primary difference between the two sets of planes is the number of bits used to store each pixel value. In the **image planes** an **8-bit byte** is used to store each RGB color hue for a total **256** available colors (intensity levels). The **overlay planes** only use **2 bits** per RGB color value for a total of **4 colors** (black, white, red, and yellow). In general the image planes are intended for full color graphics and the overlay planes work well for text.

There are three EGS Starbase viewports (Figure 2.2.5.1) for the Graphics Real-Time Program. As mentioned previously, the first is defined by device coordinates that are pixels on the display screen. They range from 0 to 1023 in the vertical and 0 to 1279 in the horizontal. These numbers are fixed by the hardware with the < 0,0 > point in the upper-left corner of the active screen area.

The second EGS Viewport has been defined around a virtual coordinate system with 0.0 to 1.0 in the vertical direction and 0.0 to 1.25 in the horizontal direction. These virtual coordinates are fixed (never change) and are used to define the overall display screen (AlphaNumeric windows). The [0.0, 0.0] point is at the lower-left of the active screen area.

The third EGS Viewport is defined in world coordinates that are in meters for the Graphics Real-Time Program. These world coordinates are only meaningful when applied to Graphics Windows (A and B). The exact zero point, size, and shape of this world coordinate system are variable in each Starbase window. They are dependent on the current **view** (map) displayed and the current pan and zoom parameters.

At any specified point in time there is always a **set of transformation matrices** between each of these three Viewports. These matrices are maintained internal to the Starbase EGS Graphics system for each of the six display windows. For each Non-Graphics window (Command, Alpha, AER, and REF) the world coordinates equal the virtual coordinates.

2.2.5.2 Window Menu Values

The WINDOW menu client display is shown in Figure 2.2.5.2 and the possible responses for each block are given in Figure 2.2.5.3.

The **first column** of boxes (except A and B Plot Axis) define the six current Starbase **Window Types**. These types are currently used for **informational** and documentation purposes, because they are fixed in the Graphics Real-Time Program (GRTP). The Type choices are F-Graphics, Alpha, Blank, Popup, P-Graphics, and P-Alpha.

HP98550A Device Coordinates

< 0, 0>
[0.0, 1.0]
(x1, y2)

<1023, 0>
[1.0, 1.0]
(x2, y2)

<1279, 0>
[1.25, 1.0]

Figure 2.2.5.1. EGS Starbase Viewports

Plot Window Type	F-Graphics	Plane	0	Rectangular Vertices	X1	0	X2	0	Y1	0	Y2	1	Y3	1
Plot Axis Types	X	X_org	Y	Y_org										
Box Window Type	F-Graphics	Plane	1	Rectangular Vertices	X1	0	X2	0	Y1	0	Y2	1	Y3	1
Plot Axis Types	X	Y_org	Y	h_msl										
Command Window Type	Popup	Plane	1	Rectangular Vertices	X1	0	X2	0	Y1	0	Y3	0.831	Y2	0.14
Alpha Window Type	Alpha	Plane	0	Rectangular Vertices	X1	1.001	X2	0	Y1	0	Y2	1.25	Y3	1
REP Window Type	Popup	Plane	1	Rectangular Vertices	X1	0	X2	0	Y1	0.88	Y2	0.2	Y3	1
REF Window Type	Popup	Plane	1	Rectangular Vertices	X1	0.56	X2	0	Y1	0.9	Y2	1	Y3	1
<div> <input type="button" value="New"/> <input type="button" value="Load"/> <input type="button" value="Save"/> <input type="button" value="Print"/> <input type="button" value="Quit"/> <input type="button" value="Help/Menu"/> </div>														

Figure 2.2.5.2. WINDOW - Graphics Window Editor

Window Types (Unused by G RTP)

F-Graphics (1) - Fixed Graphics
 Alpha (2) - Fixed AlphaNumeric (Text)
 Blank (3) - Empty (unused)
 Popup (4) - Popup (Temporary)
 P-Graphics (5) - Popup Graphics
 P-Alpha (6) - Popup AlphaNumeric

Display Planes

Image Planes (0) - 8 layer (256 colors)
 Overlay Planes (1) - 2 layer (4 colors) / Transparent

Plot Axis Types (A & B Windows)

x_org (0) - 'x' Coordinate Referenced to TTR origin
 y_org (1) - 'y' Coordinate Referenced to TTR origin
 z_org (2) - 'z' Coordinate Referenced to TTR origin
 x_ref (3) - 'x' Coordinate Referenced to Current Ref. Point
 y_ref (4) - 'y' Coordinate Referenced to Current Ref. Point
 z_ref (5) - 'z' Coordinate Referenced to Current Ref. Point
 h_msl (6) - Height (Altitude) Referenced to Mean Sea Level
 h_agl (7) - Height (Altitude) Referenced to Main Lake Target
 height (1626 meters)

Window Size & Location

Virtual Coordinates (vdc's)
 <X1, Y1> - Lower Left Corner
 <X2, Y2> - Upper Right Corner

Form (ALPHA Window)

1, 2, 3, 4, & 5
 FORM3 - Normal Air Drop & Fly Around
 FORM4 - Blank

Figure 2.2.5.3. User WINDOW Selectable Parameters

2.0 Operational Modes

The display image or overlay **Planes** are selected by the **second column** of boxes (except A and B Plot Axis). This entry designates if the window will be drawn in the **image(0)** or **overlay(1)** planes. Remember the image planes have 256 colors available while the overlay planes have 4 colors.

Starbase window size and location are set by the four boxes in the **Rectangular Vertices** entry(s). Each of these boxes takes a value that is in the **virtual coordinate** system (Figure 2.2.5.1). The coordinate pair [x1, y1] define the window's lower-left corner, while the pair [x2, y2] define the upper-right corner. To prevent distortion, the Graphics windows should be square. The other text windows must be large enough to hold the largest quantity of text (at the current font size) which may be displayed on it. If a text (Command, Alpha, etc.) window is too small, text overflow will occur.

The two Graphics windows (A and B) allow the user to select what will be plotted on the horizontal (X:) and vertical (Y:) axis. This is accomplished with the Plot Axis Types entry(s). Possible choices for each axis type are x_org, y_org, z_org, x_ref, y_ref, z_ref, h_msl, and h_agl. Some of these values(x_org, y_org, z_org, and h_msl) come directly from the MicroVax over the parallel link (pollgpio). Other values (x_ref, y_ref, z_ref, and h_agl) are calculated in the HP370, which can slow the real-time display slightly.

The ALPHA window has one extra **Form** number box, which allows the user to select the form number to display in the AlphaNumeric window. These forms are **predefined** subroutines in the **forms library** (frmllib.o). Current choices are the numbers 1, 2, 3, 4, and 5. However, only Form numbers 3 and 4 are currently available in the library; therefore, the Graphics Real-Time Program will limit the choices to 3 or 4. **FORM3** contains the normal **Air Drop and Fly Around** display parameters, while **FORM4** is a **blank** (empty) form.

2.2.5.3 Window Default File ("WINDEF")

As previously described in Section 2.2.4.2, a single startup file ("STADEF") is checked for setup parameters. One of the entries in this file is the **WINDOW File** pointer. This entry contains the **filename** of an **ASCII** disk file for the contents of this WINDOW menu. When this WINDOW client (X11) is first initialized, this WINDOW file is also used to load the default values for this menu.

A window disk file may be loaded (or saved) with Load (or Save) push button(s) at the bottom of the client menu. In order for any changes to take effect in the Real-time program, the menu information must be **explicitly saved** in the **Window File** and the current window filename must be stored in the file "STADEF". Saving the **Window File** may be done with the Save push button or at the Quit save file prompt. Initially, the **Window Default** file was named "WINDEF".

2.2.6 Radar Configuration (RADAR)

The RADAR client allows the operator to set or change the Real-time program's Radar Plot symbol values. These symbol parameters are the Marker type (plot symbol), color, and size.

2.2.6.1 Radar Selection (A, B, and all)

All Radar coordinate data is formatted in the MicroVax computer from the Target Vector Translation System (TVTS) input data streams. It is then transmitted to the HP370 computer system by the parallel I/O channel at a .1 second update rate. A Unix daemon (pollgpio) program reads the parallel input and stores the values in Target shared memory (Appendix B). The exact physical TTR radar, which is

designated Radar#1 (Radar#2, etc.), is determined by a Table located internal to the MicroVax and the current TVTS channel patching.

At the present time the HP370 Graphics Real-Time (GRTP) is capable of plotting up to eight (8) Radar Targets simultaneously (Plot All Radar mode). However, the current TVTS system can only handle five (5) TTR radars for transmission purposes. Consequently, only five Radar Targets are currently available for plotting by the Graphics Program.

The current Radar A is selected from among the five available radars at the PC Control Panel. It has a fixed Red "X" (Marker 4) plot symbol and a fixed plot size of .01 (10 pixels) in virtual coordinates (vdc's).

Radar B is also selected at the PC Control Panel from the five available radars. It has a Yellow "田" square_cross (Marker 8) plot symbol and a fixed plot size of .01 (10 pixels) in virtual coordinates (vdc's).

As mentioned, the A and B Radar plot symbols have a fixed type, color, and size. The RADAR client program menu allows the operator to select the plot symbol parameters for all the other radars when in the Plot All Radars mode.

2.2.6.2 Radar Plot Menu Values

The RADAR menu client display is shown in Figure 2.2.6.1 and the possible responses for each block are given in Figure 2.2.6.2. It is organized into eight rows where each row corresponds to a separate Radar Plot symbol. Radar#1 is the top row and Radar#8 is the last row above the push buttons.

Radar Label:	Rad1	Marker Type:	1	Marker Color Index:	Red	Marker Size:	0.01
Radar Label:	Rad2	Marker Type:	2	Marker Color Index:	Yellow	Marker Size:	0.01
Radar Label:	Rad3	Marker Type:	3	Marker Color Index:	Green	Marker Size:	0.01
Radar Label:	Rad4	Marker Type:	4	Marker Color Index:	Cyan	Marker Size:	0.01
Radar Label:	Rad5	Marker Type:	5	Marker Color Index:	Magenta	Marker Size:	0.01
Radar Label:	Rad6	Marker Type:	6	Marker Color Index:	Gray60	Marker Size:	0.01
Radar Label:	Rad7	Marker Type:	7	Marker Color Index:	Gray70	Marker Size:	0.01
Radar Label:	Rad8	Marker Type:	8	Marker Color Index:	Gray80	Marker Size:	0.01
<div> New Load Save Print Quit Help </div>							

Figure 2.2.6.1. RADAR - Radar Editor

Radar Labels (Unused by GRTP)

Radars 1 <--> 8, Starting from Top

Any four ASCII characters (Blank -> "Rad#")

Marker Type

Numerals '1', '2', '3', '4', '5', '6', '7', & '8'

Marker Numbers 10, 11, 12, 13, 14, 15, 16, & 17

Marker Color Index

Index	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	white	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0
8	10% gray	0.1	0.1	0.1
9	20% gray	0.2	0.2	0.2
10	30% gray	0.3	0.3	0.3
11	40% gray	0.4	0.4	0.4
12	50% gray	0.5	0.5	0.5
13	60% gray	0.6	0.6	0.6
14	70% gray	0.7	0.7	0.7
15	80% gray	0.8	0.8	0.8

Marker Size

0.0 to 1.0 Virtual Coordinates (vdc's)

0 to 1024 dc's (pixels)

Figure 2.2.6.2. User RADAR Selectable Parameters

The **first column** of boxes define the **eight** current **Radar Labels**. These labels are currently for informational and documentation purposes, because they are actually set in the MicroVax computer. Any **four ASCII** text characters may be entered into these boxes. If the block is left blank (empty) the characters "Rad#" will automatically be displayed. The value for "#" will depend on the corresponding row.

The plot symbol (**Marker Type**) is selected by the **second column** of boxes. This entry designates the plot symbol to use for that particular radar (Numerals '1' thru '8').

Column three contains the **Marker Color Index** and sets the corresponding plot symbol color. There are 16 possible colors to choose from.

Marker Size is set in **column four**. This plot symbol size is entered as a number whose value is between 0.0 and 1.0 (virtual device coordinates). The value 0.0 represents the smallest marker size while 1.0 represents a marker that covers the whole screen. For the HP98550a display (1024 x 1280 pixels) the value .01 is equivalent to a marker size of 10 pixels (1% of the screen size).

2.2.6.3 Radar Default File ("RADDEF")

As described in Section 2.2.4.2, a single startup file ("STADEF") is checked for setup parameters. One of the entries in this file is the **RADAR File** pointer. This entry contains the **filename** of an **ASCII** disk file, which has the contents of this RADAR menu. When this RADAR client (X11) is first started, this Radar file is also used to load the default values for this menu.

A radar disk file may be loaded (or saved) with the Load (or Save) push button(s) at the bottom left of the client menu. In order for any menu changes to take effect in the Real-time program, the menu information must be **explicitly saved** in the **RADAR File** and the current radar filename must be stored in the file "STADEF". Saving the **Radar File** may be done with the Save push button or at the Quit save file prompt. Initially, the **Radar Default** file was named "RADDEF".

2.2.7 View Configuration (VIEW)

2.2.7.1 Graphic Windows (A and B)

In the Graphics Real-Time Program (GRTP) the A and B Windows are used to display graphics maps, grids, or other drawings. Collectively these drawings are called **Views**.

Each graphics window (A or B) can have up to eight user selected Views per window. These Views are selected at the PC Control Panel. Only one View per window (A or B) may be actively displayed at a time.

Each View may consist of up to eight user selected Segments and each Segment may be turned on/off at the PC Control Panel. View and Segment construction is further discussed in Section 2.2.8.

For now this Section will describe how View files are assigned to PC Control Panel switches.

Also, window View files are loaded into the Graphics Real-Time Program when the corresponding Control Panel switch is pressed. Thus the same View file can be used for many switch numbers.

2.0 Operational Modes

Each window View (1 -> 8) has a 3 x 3 transformation matrix associated with it. These matrices are stored in Command shared memory. They are used to maintain the current View translation (pan), scale (zoom), and rotation parameters.

When the keyboard arrow keys or Previous/Next keys are depressed the current View/Window status is updated in the corresponding View matrix.

When the Home key is pressed the corresponding matrix (3 x 3) is reset to known initial values. For most cases this is the unit matrix with no translation (Offset X = 0; Y = 0), unity scale factor (1), and zero rotation. However, this initial matrix can be forced to something other than the unit matrix by declaring an Offset (X, Y), a Scale factor, and/or a View rotation angle.

2.2.7.2 View Menu Values

The VIEW menu client display is shown in Figure 2.2.7.1 and the responses for each block are given in Figure 2.2.7.2. The menu is organized into sixteen rows where the **first eight rows** refer to **Window A** and the **last eight rows** refer to **Window B**.

Each menu row corresponds to a separate View entry. Therefore, the top row corresponds to the View entry (filename) for the A Window and PC Control Panel (A) switch number one (1). Likewise, the ninth row down from the top corresponds to the View entry (filename) for the B Window and (B) switch number one (1).

The **first column** of boxes defines the View Switch Labels that are returned to the PC Control Panel when a View Names command request (GQ2) is executed. Normally, these labels are stored in the View file **header block** and they are retrieved automatically every time the **Filename** box is exited. The Get Labels push button at the bottom of the menu will also load **all** of the View Label boxes with the respective **header** values.

Different (from the header values) View Labels may be entered (4 ASCII characters) into the Label boxes if the Filename boxes are not accessed after entering new labels (or the Get Label button). This is difficult (but not impossible) because merely passing the mouse pointer (cursor) through a box can invoke the auto-update.

Switch numbers are displayed in the **second column**. They are not changeable.

The View **Filename** is entered in the **third column**. This must be a valid View (*.vfl) or Segment (*.sfl) filename and it must reside in directory "/users/egs/data".

View (Segment) filenames are checked when the box is exited and the View Label is updated. If the filename extension is omitted, the VIEW client first checks for the given name (no extension). If that file is not found, it checks for the name with a ".vfl" extension; then it checks for the name with a ".sfl" extension. The first file that exists (and it can open) is the filename that the VIEW client uses. If none of the above filenames can be found, an error message is displayed.

Empty filename boxes are not allowed. The default filename "blank.vfl" will be substituted.

Columns four, five, and six are used to enter **X and Y Offset** (translation) values. The offset Units may be in meters (M), feet (F), or nautical miles (N).

VIEW Label	Test1	Size: No	1	Filename:	HTG1A.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	Test2	Size: No	2	Filename:	HTG2A.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	RMH3	Size: No	3	Filename:	view3.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	RMH1	Size: No	4	Filename:	view1.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	RMH2	Size: No	5	Filename:	view2.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	TEST	Size: No	6	Filename:	test.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	SMAP	Size: No	7	Filename:	seemap.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	RMHP	Size: No	8	Filename:	rangemap.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	Hseg	Size: No	1	Filename:	bvren1.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	HSEG	Size: No	2	Filename:	Hseg.stl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	Null	Size: No	3	Filename:	null.stl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	NULL	Size: No	4	Filename:	blank.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	NULL	Size: No	5	Filename:	blank.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	Null	Size: No	6	Filename:	null.stl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	SHGT	Size: Yes	7	Filename:	sechgt.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0
VIEW Label	PHGT	Size: No	8	Filename:	rangehgt.vfl	Offset: No	0	Units:	0	Scale:	M	Rotate:	1	0

Figure 2.2.7.1. VIEW - View Editor

Top eight lines, A Window Views 1 -> 8

Lower eight lines, B Window Views 1 -> 8

View Labels

Individually read & updated when the 'Filename' block is exited. (File header information).

Button 'Get Labels' updates all View Labels.

Labels passed to PC Control Panel.

Switch Numbers

'1', '2', '3', '4', '5', '6', '7', '8'

Filenames (*.vfl or *.sfl)

Segment/ View filenames to be used with the respective Switch Numbers.

Directory '/users/egs/data'

'View' filenames are checked when the block is exited and the 'View' Label is updated.

Empty filename blocks are not allowed. Filled with "blank.vfl".

Offset X & Y ; Units (Pan)

'View' offset from TTR Range zero in the indicated Units.

Units - Meters(M), Feet(F), or Nautical Miles(N)

Figure 2.2.7.2. User VIEW Selectable Parameters

Scale (Zoom)

Scale Factor applied to View Window.

Referenced to TTR Range zero.

No Change -> Scale Factor = 1.0

Zoom In -> Scale Factor > 1.0

Zoom Out -> Scale Factor < 1.0

(Negative -> Reciprocal , -2 ~ .5)

Rotate (Confusing)

Rotate View by X number of degrees

Referenced to TTR Range zero.

North - 0 degrees

Clockwise -> +degrees

Counterclockwise -> -degrees

Figure 2.2.7.2. User VIEW Selectable Parameters

2.0 Operational Modes

This is the offset from the TTR range zero point. Normally, it will tend to bring the designated point toward the center of the display window. However, it also depends on several other factors like the current View world coordinate window definition. Experimenting with the different offsets is the best advice, especially when Scale and Rotation are also used.

The View **Scale** factor is entered in **column seven**. Again the scale factor change (magnification) is applied centered at the TTR range zero point. This is also the world coordinate (0.0, 0.0) point.

All world coordinate window primitives (not device coordinates) are scaled up/down according to this value. A scale factor of one (1) implies no change in the drawing. A scale factor greater than one (>1) causes all primitives to be drawn larger (Zoom In). Scale factors less than one (<1) or negative cause the primitives to be drawn smaller (Zoom Out). Note a scale factor of zero (0) is not accepted (converted to 1).

Column eight contains the View **Rotate** entry box. This rotation is performed around the TTR range zero point. The rotate value is entered in degrees with zero (0) degrees referenced to North. Plus (+) degrees indicates a clockwise rotation, while minus (-) degrees causes a counterclockwise rotation of the drawing primitives. It only effects primitives drawn with world coordinates.

2.2.7.3 View Default File ("VEWDEF")

In Section 2.2.4.2 a single startup file ("STADEF") is checked for setup parameters. One of the entries is the **VIEW** File pointer. This entry contains the **filename** of an **ASCII** disk file, which has a copy of the contents of this VIEW menu. When this VIEW client (X11) is first started, this menu file is also used to load the default values for this menu.

A menu disk file may be loaded (or saved) with the Load (or Save) push button(s) at the bottom left of the client menu. In order for any menu changes to take effect in the Real-time program, the menu information must be **explicitly saved** in the menu file and the current **VIEW File** pointer must be stored in the file "STADEF". Saving the menu file may be done with the Save push button or at the Quit save file prompt. Initially, the **menu default** file was named "VEWDEF".

2.2.8 Segment/View Editing

As previously mentioned in Section 2.2.7.1, the Graphics Real-Time Program (GRTP) uses the A and B windows to display graphics maps, grids, and other drawings. These drawings primitives are collectively organized into Views. The X11 client View is used to associate View (or Segment) filenames with the PC Control Panel switches.

Each graphics window (A or B) can have up to eight user-selected Views per window. These Views are selected at the PC Control Panel and only one View per window (A or B) may be actively displayed at a time.

Each EGS View may consist of up to eight user-selected Segments and each Segment may be turned on/off at the PC Control Panel. A Segment is a collection of drawing primitives which together make up a picture.

The EGS Starbase/Display List View files (*.vfl) have a hierarchy as shown in Figure 2.2.8. They consist of a **Main Segment** whose purpose is to **call and execute** other Starbase **Segments** (*.sfl). The Main Segment is the Display List execution coordinator.

Starbase/Display List View

Figure 2.2.8. HP EGS Views and Segments

Although the primary purpose of the Main Segment is as described above, any valid **Segment** file (*.sfl) may be loaded in its place. The limitation to this substitution is that the hierarchy may not be implemented and the sub-segment filter (on/off function) may not work.

The **Main Segment** in a View can never be turned off by the **Segment** switches of the PC Control Panel. However, a new **View** can be loaded by the PC **View** switches.

2.2.8.1 EGS Starbase File Types

The EGS Graphics files may be stored in any of five file formats. These file types are as follows:

1. ASCII File Format (*.asc)
2. Segment File Format (*.sfl)
3. View File Format (*.vfl)
4. Bitmap Image Format
5. Command File Format (*.com) - XREF

Appendix E contains a detailed description (with examples) of the ASCII, Segment, View, and Bitmap Image formats. A brief description of each of these formats follows.

The **ASCII File** format (*.asc) is a direct text representation of the Starbase/Display List graphics commands. It is the simplest format for the user to read and interpret. Also, it is the most useful when transferring drawings between graphics systems or when creating/editing EGS segments in a non-Starbase application program ('vi', TCALL, WAYPOINTS). Appendix H contains a detailed description of how to create and build View files (*.vfl) from ASCII files (*.asc).

2.0 Operational Modes

The Display List **Segment File** format (*.sfl) is analogous to a compiled version of the ASCII file format. It can be loaded and displayed (traversed) by any EGS Starbase application program. The Segment format is a binary (integer) file that is defined in the Display List library and it can be edited (GRTP) using Display List commands. However, it is not easily read by a user. Client program SHOW_SEG (ConvMap) will read an ASCII file (*.asc) and produce a Segment file (*.sfl) as an output.

The Display List **View File** format (*.vfl) is identical to the Segment format with the following changes. A Main Segment is created and the Segment hierarchy is fully implemented. **Note:** All Segment files are copied to the View file; thus the binary View file is a complete entity by itself.

The X11 client program BUILD (ViewBld) creates/edits the View file (*.vfl) from either ASCII files (*.asc) or Segment files (*.sfl).

The **Bitmap Image** format is used to store a frame buffer or memory image of the screen. This format is used by the EGS Graphics Real-Time Program to store screen views for subsequent spooled output to the line printer. It is not used for any other purpose.

Command File format is a subset of the old **Tektronix 4081** file format. The Tektronix XREF command utility used these commands when creating a drawing on the Tektronix display. These files must have a ".com" file extension. The X11 client program SHOW_SEG can convert this file format to the new ASCII format (*.asc) and the Segment format (*.sfl). This Command File format is included to provide backward compatibility with older Tektronix type files.

2.2.8.2 General View/Segment Procedure

The first step in constructing a drawing for use by the Graphics Real-Time Program is to create the **ASCII file** (*.asc). This file contains the Starbase graphics commands and it is in a user-readable text format. This ASCII file may be created in several ways, but the most general is to use the UNIX 'vi' text editor (Appendix H). Other ways to create this text file are shown in Figure 2.2.8.1 and include the client programs **TCALL** and **WAYPOINTS**. For now the discussion will be limited to the 'vi' editor.

Note: All files mentioned in this Section are assumed to reside in directory '/users/egs/data'. By default all X11 Setup client programs look for an input filename in this directory and save all output files to this directory.

The standard UNIX 'vi' editor can be executed from any X11 terminal emulator window. One such client window (login) is available in the EGS X11 Setup Mode (Figure 2.2.2). To start the text editor, the operator must have a valid password and successfully log into the HP UNIX system. The **login** client is a full hp terminal emulator (hpterm) window and the user has all the capabilities which his password allows.

To create/edit a file, type 'vi' and the filename (*.asc). Use the edit commands to modify the text file contents and then save the file to disk. Finally, exit the editor program. **Note:** This same 'vi' editor can be used to create/edit Command files (*.com).

After the **ASCII file** (*.asc) is created (or edited), it can be converted to the **Segment** integer format (*.sfl) by the X11 client program **SHOW_SEG**. This Segment editor is discussed more fully in Section 2.2.8.3.

Figure 2.2.8.1. Building EGS Segments and Views

2.0 Operational Modes

Once the drawing Segments exist, they are combined to form the display Views by the View editor **BUILD**. This X11 Setup client creates the hierarchy of the Display List file (*.vfl) and is described in Section 2.2.8.4.

Both the **Segment (*.sfl)** and **View (*.vfl)** files are stored in the **integer format**. This integer format allows a quicker file loading by the Real-time program.

Finally, the newly created/edited **View** file is associated with a **PC Control Panel** switch by the X11 client **VIEW** (see Section 2.2.7).

2.2.8.3 Segment Editor (SHOW_SEG)

The X11 Setup client program **SHOW_SEG** is an extremely versatile **Segment Editor** drawing format converter. It allows the user to display any Segment file (*.asc, *.com, or *.sfl) in a separate X11 window **SegWin**. Also, it converts the different file formats as follows:

1. ASCII (*.asc) → ASCII(*.asc) & Segment (*.sfl) Integer
2. Command (*.com) → ASCII(*.asc) & Segment (*.sfl) Integer
3. Segment (*.sfl) → Display Only

In addition, the **SHOW_SEG** client allows the user to edit a file's header information. It is much safer to edit the header data in this client program than with the UNIX 'vi' editor commands. Finally, the user can change the current Segment's world coordinate window values with this client. The new window values may be typed into the **Window Coordinate** menu blocks or they may be changed **interactively** with the **mouse**.

Figure 2.2.8.2 shows the **SHOW_SEG** menu display and Figure 2.2.8.3 shows an example 'SegWin' display.

Figure 2.2.8.2. **SHOW_SEG - Segment Display Editor**

Figure 2.2.8.3. X11 Display "SegWin" Window

2.0 Operational Modes

A utility program called "ConvMap" performs a similar function to SHOW_SEG but it only executes outside of the X-Window server environment. It is further described in Section 3.1.

2.2.8.3.1 SHOW_SEG Menu Values

The **top row** in the SHOW_SEG menu is used to define the **Input** (*.asc) and **Output** (*.sfl) filenames. The first block (column) is used to enter the Segment Input filename. By default, the SHOW_SEG client looks for input files in directory '/users/egs/data'.

The Input extension type (*.asc, *.com, or *.sfl) determines what format conversion will be performed on the file. If no extension is present a **Command** file (*.com) type is assumed. The SHOW_SEG client first checks for the named Input file; if it cannot be found an error message is displayed in the Console Window. This check is performed when the **SHOW_SEG** is selected.

Note at certain times (edit mode) both ASCII (*.asc) and Segment (*.sfl) files are saved (overwritten). If a file with the same name already exists, the user will be prompted to allow overwriting.

The second block (column) is used to enter the Output Segment (*.sfl) integer filename. By default, this is always forced to the **root** ASCII (or Command) **name** with an extension of **".sfl"**. This filename is stored in the file header information block (application_data).

User comments may be entered in the **second row** in the box labeled **Remarks**. A maximum of 80 text characters may be entered. These remarks are subsequently stored inside the file with the header data.

The **third row** consists of two entries, the **Switch Number** and **Segment Name**. Most of the time these entries are not used. The Switch Number is actually assigned in the VIEW client program. However, this switch number entry is stored in the Segment file's header and would be available for future uses.

The Segment Name is only used if this Segment file is loaded as the main segment. Then it will be the default switch label for the PC Control Panel. The Segment Name is stored in each Segment's file header and is a maximum of four (4) text characters long. It may be blank (empty).

The **Author's** name is an optional entry in the **fourth row**. It may be up to 20 characters long and is saved in the header block.

The **Window Units** block (**fifth row**) defines the measurement units related to the Window Coordinates on the next two rows. Acceptable unit values are meters (M), feet (F), or nautical miles (N). However, when the Segments are drawn, and stored to disk, all coordinates are converted to meters.

Display **Window Coordinates** are entered on **rows six and seven**. These are the EGS Starbase world coordinates (extremes) of the display view window (SegWin). They are stored in the Segment files (*.asc and *.sfl) as the Starbase "view_window" graphics primitive entry. The two pairs (<x1,y1> and <x2,y2>) define the diagonal vertices of a rectangle in the world coordinate system and are referenced to the TTR range zero point. The user may enter any set of coordinate pairs but it is his responsibility to check for a valid window ($x1 < x2$ and $y1 < y2$) on any values that are typed in these boxes.

Another way to change **Window Coordinates** is to press the **Window** push button. This function allows the operator to use the mouse locator input with the current graphics window (SegWin) when defining

the window coordinates. The newly selected coordinate pairs <x,y> will appear in the Window Coordinate boxes (Units - meters) after this operation completes.

Note the new coordinates will not be permanently saved (or drawn) until the **SHOW_SEG** push button is selected.

2.2.8.3.2 SHOW_SEG Functions

The first three push buttons, on the lower left of the Menu, are the screen Edit buttons. They perform a different function from the previously described clients (START, WINDOW, RADAR, and VIEW). The values saved in this file are still the parameters appearing on the screen menu. As such they can be cleared (**New**), loaded from disk (**Load**), and stored to disk (**Save**). The default screen filename is "SEGDEF". However, these screen menu files have nothing to do with the Real-Time Program (GRTP). The useful output from this client is stored in the Output files (*.asc and *.sfl). Screen menu files (i.e., "SEGDEF") may be used as temporary scratch files, to save a menu setup, while performing another task (Windowing). Also, this file's contents determine the default menu values when the client is started.

The **Seg_Info** push button is a client Specific screen Edit button. It reads the **file header information** and displays it in the menu. The **Seg_Info** operation is used **only** when editing an existing file. The filename currently entered in the Input file block is used when searching for this header and its file extension will be forced to the ASCII type (*.asc).

Push button **Window** is another client Specific screen Edit button. This function allows the operator to use the mouse locator input with the graphics window (SegWin) when defining the window coordinates. The newly selected rectangle coordinate pairs <x,y> are placed in the Window Coordinate boxes when this function completes. Also, the new coordinates are not permanently saved (or drawn) until the **SHOW_SEG** push button is selected.

The **Window** function asks the user if they would like to use a Section Map window before starting the window operation. If the answer is Yes, the Section Map coordinates (-259000, -315000 and 301000, 245000) are entered into the World Coordinate Edit Boxes. However, the graphics window (SegWin) will not be re-drawn until the **SHOW_SEG** push button is pressed.

Also, the rectangular window created by the mouse locator will always represent a square (isometric) when returned as **Window Coordinates**. This square's sides are equal to the largest dimension of the mouse's rectangle. It is centered, as much as possible, on the mouse's rectangle center point.

The **Print** push button lists the current contents of the screen menu to the line printer.

SHOW_SEG is a client specific Execute push button. Selecting this button actually performs the client function. In this case, it reads the Input file, converts formats, and displays the Segment in the graphics window (SegWin).

The **Quit** button halts this client program's execution, removes the graphics window (SegWin), cleans up memory references, and removes the process from the process list. It is **permanent** in this session with the X Server. If the user is not sure whether this client may be needed again, minimize (icon) both the **SHOW_SEG** and graphics (SegWin) windows.

Button **Help Me!** causes a pop-up window to appear with any available help information.

2.2.8.4 Build View Editor (BUILD)

The X11 Setup client program **BUILD** is a general-purpose EGS **View Editor**. It will construct the EGS Main calling segment and the **Segment hierarchy**. It allows the user to display any View file (*.vfl) in a separate X11 window **ViewWin**. Also, it converts the ASCII file format (*.asc) or Segment (*.sfl) into the View integer format (*.vfl). The output View file (*.vfl) is a complete entity unto itself and does not require any further access to any additional files.

In addition, the BUILD client allows the user to create/edit a separate header for the View file. This header information is completely separate and distinct from the header in each Segment file.

Finally, the user can select/change the current View's world coordinate window values with this client. The new window values may be typed into the **Window Coordinate** menu blocks or they may be changed **interactively** with the **mouse**. These **View** window coordinates supersede any component Segment values.

Figure 2.2.8.4 shows the BUILD menu display and Figure 2.2.8.5 shows an example 'ViewWin' display.

A utility program called "ViewBld" performs a similar function to BUILD but it only executes outside of the X-Window server environment (Section 3.1).

2.2.8.4.1 BUILD Menu Values

The top seven rows of the BUILD menu are identical to the first seven rows of the SHOW_SEG client program. Also, they perform similar types of operations. The main difference is that SHOW_SEG operates on Segments (*.sfl) while BUILD operates on Views (*.vfl).

Input (*.asc) and **Output** (*.vfl) filenames are defined in the **top row** of the BUILD menu. The first block is used to enter the View Input filename editing an existing file. In addition, this ASCII (*.asc) file is created on output the first time the View file is built. The Input/Output ASCII filename extension is forced to ".asc". By default, the BUILD client looks for input files in directory '/users/egs/ data'.

If the Input file (*.asc) cannot be found an error message is displayed on the Console Window. This check is performed when the **BUILD** is selected.

At certain times (edit mode) both ASCII (*.asc) and View (*.vfl) files are saved (overwritten). If a file with the same name already exists, the user will be prompted to allow overwriting.

Note that the ASCII file (*.asc) is only a representation of the main segment with header information, view window definition, filtering information, and Segment calls. In addition, the View integer file (*.vfl) contains all of the main graphics calls and a copy of every segment file (integer - *.sfl) that makes up the view.

The second block (column) is used to enter the Output View (*.vfl) integer filename. By default, this is **always** forced to the **root ASCII name** with an extension of ".vfl". This filename is stored in the file header information block (application_data).

User comments may be entered in the **second row** in the box labeled **Remarks**. A maximum of 80 text characters may be entered. These remarks are subsequently stored inside the file with the header data.

View Filename:

Remarks:

Switch Number: View Name:

Author:

Window Units:

Window Coordinates (Lower, left):

Upper, right:

Segment #1 name:

Segment #2 name:

Segment #3 name:

Segment #4 name:

Segment #5 name:

Segment #6 name:

Segment #7 name:

Segment #8 name:

Figure 2.2.8.4. BUILD - View Build Editor

Figure 2.2.8.5. X11 Display "ViewWin" Window

The **third row** consists of two entries, the **Switch Number** and **View Name**. Switch Number is not used by the Real-time program but it is stored in the file header. The actual Switch Number is assigned in the VIEW client program.

View Name is the default switch label for the VIEW client's **View Label's** entry and subsequently the switch label for the PC Control Panel. The View Name is stored in each View's file header and is a maximum of four (4) text characters long. It may be blank (empty).

The **Author's** name is an optional entry in the **fourth row**. It may be up to 20 characters long and is saved in the header block.

The **Window Units** block (**fifth row**) defines the measurement units related to the Window Coordinates on the next two rows. Acceptable unit values are meters (M), feet (F), or nautical miles (N). However, when the Views are drawn, and stored to disk, all coordinates are converted to meters.

Display **Window Coordinates** are entered on **rows six and seven**. These are the EGS Starbase world coordinates (extremes) of the display view window (ViewWin). They are stored in the View files (*.asc and *.vfl) as the Starbase "view_window" graphics primitive entry. The two pairs (<x1,y1> and <x2,y2>) define the diagonal vertices of a rectangle in the world coordinate system and are referenced to the TTR range zero point. The user may enter any set of coordinate pairs but it is their responsibility to check for a valid window ($x1 < x2$ and $y1 < y2$).

Another way to define **Window Coordinates** is to press the **Window** push button. This function allows the operator to use the mouse locator input with the current graphics window (ViewWin) when defining the window coordinates. The newly selected coordinate pairs <x,y> will appear in the Window Coordinate boxes (Units - meters) after this operation completes.

Note the new coordinates will not be permanently saved (or drawn) until the **BUILD** push button is selected.

2.2.8.4.2 BUILD Functions

The first three push buttons, on the lower left of the Menu, are the screen Edit buttons. They perform a different function from the previously described clients (START, WINDOW, RADAR, and VIEW). The values saved in this file are still the parameters appearing on the screen menu. As such they can be cleared (**New**), loaded from disk (**Load**), and stored to disk (**Save**). The default screen filename is "BLDDEF". However, these screen menu files have nothing to do with the Real-Time Program (GRTP). The useful output from this client is stored in the Output files (*.asc and *.vfl). Screen menu files (i.e., "BLDDEF") may be used as temporary scratch files, to save a menu setup, while performing another task (Windowing). Also, this file's contents determine the default menu values when the client is started.

The **Get_Segments** push button is a client Specific screen Edit button. It reads the **file header information** and displays it in the menu. Also, it reads the currently selected **Segment filenames** and displays them on the menu. The **Get_Segments** operation is used **only** when editing an existing file. The filename currently entered in the Input file block is used when searching for this header (filenames) and its file extension will be forced to the ASCII type (*.asc).

2.0 Operational Modes

Push button **Window** is another client Specific screen Edit button. This function allows the operator to use the mouse locator input with the graphics window (ViewWin) when defining the window coordinates. The newly selected rectangle coordinate pairs <x,y> are placed in the Window Coordinate boxes when this function completes. Also, the new coordinates are not permanently saved (or drawn) until the **BUILD** push button is selected.

The **Window** function asks the user if they would like to use a Section Map window before starting the window operation. If the answer is Yes, the Section Map coordinates (-259000, -315000 and 301000, 245000) are entered into the Window Coordinates Edit Boxes. However, the graphics window (ViewWin) will not be redrawn until the **BUILD** push button is pressed.

Also, the rectangular window created by the mouse locator will always represent a square (isometric) when returned as **Window Coordinates**. This square's sides are equal to the largest dimension of the mouse's rectangle. It is centered, as much as possible, on the mouse's rectangle center point.

The **Print** push button lists the current contents of the screen menu to the line printer.

BUILD is a client specific Execute push button. Selecting this button actually performs the client function. In this case, it reads the Input file, converts formats, builds the hierarchy, and displays the View in the graphics window (ViewWin).

The **Quit** button halts this client program's execution, removes the graphics window (ViewWin), cleans up memory references, and removes the process from the process list. It is **permanent** in this session with the X Server. If the user is not sure whether this client may be needed again, minimize (icon) both the **BUILD** and graphics (ViewWin) windows.

Button **Help Me!** causes a pop-up window to appear with any available help information.

2.2.9 WayPoint Editor (WAYPOINT)

Two other methods exist to construct specialized EGS Segments besides the 'vi' editor. They are the WayPoints Editor and the Air Drop Editor (TCALL - Section 2.2.10). Both of these specialized Segment editors prompt the users for information and then produce ASCII Segment files (*.asc) as outputs. The ASCII file output means that the WAYPOINT (TCALL) processes do not require that the Starbase/Display List library be attached.

2.2.9.1 WayPoint Text Files

The input data to the WayPoint Editor process originates from two sources, the screen menu and a WayPoint text file (*.txt).

A **WayPoint** is a coordinate point <x,y,z> in space (range) toward which an aircraft (missile, etc.) is flying. It may reach this WayPoint coordinate and then proceed to the next WayPoint. Another, more realistic, concept is that the aircraft never quite reaches the current WayPoint. Instead it follows a curved trajectory (turn radius) when proceeding to the next WayPoint coordinate. In the **WAYPOINT** Editor process the WayPoint coordinates are entered as Latitude, Longitude, and Height values.

Subroutine GEAZ converts the global coordinates into TTR range coordinates <x,y,z> in meters. Also, note the Longitude at Tonopah Test Range is entered as a positive value when it is in fact negative in the World Coordinate system. This fact is understood and it is accounted for in the GEAZ calculations.

A **WayPoint Text file** (*.txt) consists of a series of consecutive WayPoints which, when taken together, form a **flight profile** for an aircraft to follow during a test. The Text file consists of one line per WayPoint coordinate. An example Text file is shown in Figure 2.2.9.1.

The **first** entry in each line is the **WayPoint number**. WayPoint numbers may start and stop at any arbitrary value. However, they must be monotonically increasing. This means that the **first** WayPoint number must be **less than** the last WayPoint number. Also, each successive number must be greater (larger) numerically than the previous number. **Note:** It is acceptable if the WayPoint numbers are not consecutive (values may be skipped).

Entry **two** in each line is the **WayPoint Latitude** value. Latitude may be entered in two formats.

DMS - Degrees Minutes Seconds (37 50 51)

dEC - Decimal (37.8475)

Entry **three** in each line is the **WayPoint Longitude** value and uses the same format as the Latitude value.

The **fourth** entry in a line is the **Height** parameter and may be in feet (F) or meters (M).

The **last** entry on each line is the **Turn Radius** in degrees. It is an optional entry and if it is absent a Turn Radius of 0.5 degrees (default) is assumed. Note a Turn Radius of 0 degrees is an acceptable entry and it will cause straight line segments to be drawn between WayPoints.

The Turn Radius calculation is somewhat tedious to follow and is further explained in Appendix K. Basically the WayPoint trajectory is drawn by looking at three consecutive WayPoints (previous, current, and next). A **constant radius arc** is drawn connecting WayPoint lines from the previous coordinate to the next coordinate at the current WayPoint location. This constant radius arc is calculated based on one nautical mile of radius per degree of Turn Radius (1 Nmile/degree).

2.2.9.2 WayPoint Menu Values

The X11 Setup client program **WAYPOINTS** takes WayPoint coordinates from a WayPoint Text file (*.txt) and constructs three ASCII Segment Output files (*.asc). The operator may then take these Segment files and use them to build Views (BUILD and VIEW) for the Graphics Real-Time Program (GRTP).

Information displayed in the WAYPOINTS Menu is used to construct these files and it is stored in each file's header.

Three ASCII Output Segment files (*.asc) are produced by this client program. This allows the operator to separately display the course path, the WayPoint markers, or the marker annotation text.

The **first** Segment file contains the **lines** and **curves** used to construct the predicted course trajectory. It is called the **WayPoints (waypts.asc)** file.

WayPoint				
#	Latitude	Longitude	Hght	Turn Radius
0	37 50 51	116 42 29	5345	5
1	37 37 31	116 29 59	6800	5
2	37 29 40	116 22 39	6500	5
3	37 22 48	116 28 23	6300	5
4	37 24 47	116 35 32	6000	5
5	37 33 20	116 41 22	6600	5
6	37 38 35	116 47 31	7200	5
7	37 41 02	116 50 17	6900	5
8	37 41 17	116 56 03	6000	5
9	37 38 25	116 59 23	5200	5
10	37 35 11	117 00 06	6200	5
11	37 08 16	116 46 58	7000	5
12	37 06 30	116 38 19	7000	5
13	37 12 07	116 35 20	7000	5
14	37 27 47	116 34 48	6200	5
15	37 33 26	116 41 28	6500	5
16	37 33 50	116 41 47	6600	5
17	37 38 40	116 47 37	7000	5
18	37 40 48	116 50 26	6800	5
19	37 41 28	116 55 59	6000	5
20	37 38 04	116 59 27	6000	5
21	37 34 43	116 59 57	6500	5
22	37 07 45	116 46 44	7000	5
23	37 05 35	116 38 01	7000	5
24	37 11 22	116 35 32	7000	5
25	37 27 54	116 34 35	6800	5
26	37 32 36	116 40 21	6800	5
27	37 33 46	116 41 52	6500	5
28	37 38 35	116 47 31	6800	5
29	37 40 38	116 49 51	6600	5
30	37 41 42	116 55 59	6000	5

Figure 2.2.9.1. 'd13tra.txt' WayPoint Text File

Markers (circles) are output to a **second** Segment file at each WayPoint coordinate location. The size of each marker is selected to be approximately two percent (2%) of the initial window size. This file is called the **Marker (marker.asc)** Segment file.

The **last (third)** Segment file contains the **marker numbers** in text format that are displayed at each WayPoint coordinate location. These are the exact WayPoint numbers as they appeared in the input text file (*.txt). It is called **Text Annotation (annotate.asc)** Segment file.

The first three menu blocks in the WAYPOINTS menu window (Figure 2.2.9.2) contain test specific information. This test data is stored in the Command shared memory block and is identical to the Test Information described in Section 2.2.4.1. In fact, changing (and saving) the Test Information in this menu will cause the corresponding data in the START menu (and TCALL menu) to change.

The **Test ID** Edit block can contain any combination of up to 30 ASCII characters.

The **Operation no.** block may consist of a maximum of 12 ASCII characters.

The **Test Date** menu block has room for 8 ASCII characters in a free-field format.

The **Author's name** is an optional entry in the **fourth row**. It may be up to 19 characters long and is saved in the header blocks of all three files (*.asc).

User comments may be entered in the **fifth row** in the block labeled **Comments**. A maximum of 40 text characters may be entered. These remarks are subsequently stored inside the files with the header data.

Row six contains the **WayPoint Input filename** block and its defining parameters. The filename can be 15 characters long and its contents must be as shown in Figure 2.2.9.1. The file's default extension type is '.txt' and it must exist in directory '/users/egs/data'. If the named file does not exist an error message will be displayed.

The **second entry** in row six is the input file **Format**, which is the format of the Latitude and Longitude values as follows:

DMS - Degrees Minutes Seconds (116 42 29)

dEC - Decimal (116.70806)

Two Input file **Type's** (column three) are accepted by the WAYPOINTS program. They are differentiated by the number of entries on each line of the input file.

NORMAL - Latitude Longitude Height

WAYPOINT - Latitude Longitude Height Radius

Note: If **Turn Radius** is absent a default Turn Radius of **0.5 degrees** is used by the WAYPOINT program.

The input file's **World Coordinate System** is identified by the **System** (column four) block. At the present time, two World Coordinate datums are recognized by the WAYPOINTS program.

Test ID												
Operation no.	R123456											
Test Date	06/07/93											
Author's name	Ken Bauhs											
Comments	Test of file 'd13tra.txt'											
Waypoint INPUT file name	d13tra.txt	Format	DMS	Type	WAYPOINT	System	NAD	Units	F	Height point/Hs10	M	
OUTPUT FILE names (evnts.asc)	waypts.asc		marker.asc		waymrk.asc		(annotate.asc)		wayano.asc			
OUTPUT Segment names (evnts)	wayp		(marker)		mark		(annotate)		anno			
Start point no.	0	Stop point no.	28									
NEW	Load	Save	Print	FILE points	CEP	ICHP	Quit	Help				

Figure 2.2.9.2. WAYPOINTS - WayPoints Calculator

WGS-84 - WGS 84 Coordinates

NAD - NAD 1927 Coordinates

The **last two blocks** (columns five and six) define the input file's **height** parameter. The **Units** block can be either feet (F) or meters (M). Finally, the **H(Ellipsoid/msl)** block indicates the Height Reference point as follows:

E - Height value is referenced to the current Ellipsoid surface.

M - Height value is referenced to mean sea level (msl).

Rows seven and eight define the three **Segment Output filenames**. The three output files are the WayPoints file, the Marker file, and the Annotation file. All files are stored in directory '/users/egs/data' and have the default extension '.asc' for ASCII files. If the output files do not exist, they are created. If they already exist, they are overwritten with new data. **Note:** No overwrite prompt is displayed. The filenames may be 15 characters long (maximum).

waypts.asc - Arc and Line Segments

marker.asc - Circular markers at each WayPoint

annotation.asc - 'Text' WayPoint Numbers

Figures 2.2.9.3, 2.2.9.4, and 2.2.9.5 show examples of the three ASCII Segment Output files as displayed by the SHOW_SEG client program.

The **OUTPUT Segment names** are only used if a Segment file is loaded as a main Segment. Then it will be the default switch label(s) for the PC Control Panel. The Segment Names are stored in each Segment's file header and are a maximum of four (4) text characters long. It may be blank (empty).

Row nine (last row above push buttons) allows the operator to enter **Start and Stop WayPoint numbers**. These are the actual WayPoint numbers as entered in the WayPoint Input file. They allow the operator to declare the WayPoint number to begin and end processing in the TOMAH function. It allows successive runs of the WAYPOINT program with the same input file (*.txt) but different WayPoint paths and output file Segments (*.asc). In turn the different output Segments may be loaded into separate Views for display in the Real-time program.

The WayPoint numbers may start and stop at any arbitrary values in the input file. However, the numbers must be monotonically increasing. This means that the **Start** WayPoint number must be **less** than the **Stop** WayPoint number.

2.2.9.3 WayPoint Functions

The first three push buttons, on the lower left of the WAYPOINTS Menu, are the screen Edit buttons. The values saved in this file "WAYDEF" are the parameters appearing on the screen menu with the exception of the Test ID, Operation no., and Test Date blocks. These three blocks are saved in Command shared memory and the START default file "STADEF". Other screen values can be cleared (**New**),

Figure 2.2.9.3. 'waypts.asc' Segment File - Arc and Line Segments

Figure 2.2.9.4. 'marker.asc'- Circular Markers at Each WayPoint

Figure 2.2.9.5. 'annotate.asc' - Text WayPoint Numbers

loaded from disk (**Load**), and stored to disk (**Save**). **Note:** The Save push button performs a complete save including the first three parameters, Test ID, Operation no., and Test Date to "STADEF".

These screen menu files have little to do with the Graphics Real-Time Program (GRTP). The useful output from this client is stored in the three Output files: waypts.asc, marker.asc, and annotate.asc. Screen menu files (i.e., "WAYDEF") may be used as temporary scratch files, to save a menu setup, while performing another task. Also, this file's contents determine the default menu values when the client is started.

The **Print** push button lists the current contents of the WAYPOINTS menu to the line printer.

FILE_points is a client specific Execute push button. Selecting this button reads the WayPoint Input file (*.txt) and writes the complete WayPoint listing to the line printer in the format of the Input file.

The **GEAZ** push button Executes a client function that reads the Input file (*.txt), converts World Coordinates to TTR range coordinates <x,y,z>, and saves the results in a memory array. As an option, the converted TTR range values may also be written to a line printer. All WayPoints in the Input file (*.txt) are converted in one pass through the GEAZ function.

TOMAH is another client specific Execute push button. Selecting this button actually performs the WAYPOINTS client function. In this case, it takes the memory array values (GEAZ) and constructs the Output ASCII Segment files: waypts.asc, marker.asc, and annotate.asc.

Note that function GEAZ must have been run (at least once) before TOMAH can be executed. If TOMAH detects that GEAZ has not been run, it will call and execute it (silently) for you. However, if GEAZ has been run at least once, manually or silently by TOMAH, and the Input filename (*.txt) is changed the user must select the GEAZ button to input/convert the new values.

Menu values for **Start** and **Stop point numbers** are used by the **TOMAH** function when constructing **Output Segment files** (*.asc). The TOMAH function can be run as many times as necessary, with different Start/Stop numbers and Output filenames, to obtain the required path/trajectory Views. However, the **GEAZ** function needs to be executed only once per **Input file** (*.txt) selection.

The **Quit** button halts this client program's execution, cleans up memory references, and removes the process from the process list. It is permanent in this session with the X Server. If the user is not sure whether this client may be needed again, minimize (icon) the WAYPOINTS window.

Button **Help Me!** causes a pop-up window to appear with any available help information.

2.2.10 Air Drop Editor (TCALL)

Another specialized Segment editor, besides WAYPOINTS (Section 2.2.9), is the Air Drop Editor (TCALL). TCALL prompts the user for specific information, by the X11 menu window, and produces two ASCII Segment files (*.asc) as outputs. The ASCII file output means that the TCALL process does not require that the Starbase/Display List library be attached to this client program.

2.2.10.1 TCALL Menu Values

The X11 Setup client program **TCALL** takes information from the TCALL Menu window and constructs two ASCII Segment Output files (*.asc). Views may then be built (BUILD and VIEW) for the Graphics

2.0 Operational Modes

Real-Time Program (GRTP), which allows the operator to separately display the course line and the altitude plots.

The **first** Segment file contains the course run-in line (x and y) with time tick marks from 0 seconds (release point) to 360 seconds (6 minutes) before release. It is designated as the **GENCAL** Segment (*.asc) file.

Segment file number **two** is an altitude versus distance (time) grid plot for the bomb release countdown. It contains both time grid lines and height boundaries for the release trajectory. This file is called the **HVSY** Segment (*.asc) file.

Figure 2.2.10.1 shows the TCALL X-window Menu display.

The first three menu blocks in the TCALL window contain test-specific information. This test data is stored in the Command shared memory block and is identical to the Test Information described in Section 2.2.4.1. In fact, changing (and saving) the Test Information in this menu will cause the corresponding data in the START menu (and WAYPOINTS menu) to change.

The **Test ID** Edit block can contain any combination of up to 30 ASCII characters.

The **Operation no.** block may consist of a maximum of 12 ASCII characters.

The **Test Date** menu block has room for 8 ASCII characters in a free-field format.

Menu lines **four** and **five** are used to enter the **Target Name** and **Target location** in TTR range coordinates. The Target name may be up to 10 ASCII characters long and appears at the Target point in the GENCAL segment (run-in line) plot. Target coordinates (x, y, h) may be entered in meters (M), feet (F), or nautical miles (N).

Row **six** consists of blocks for entering **wind** correction factors (X and Y). These correction factors may be entered in units of meters (M), feet (F), or nautical miles (N). The wind correction factors (x and y) are directly subtracted from all calculated distance (time) values from the release point back. However, the Target point remains unchanged. This causes a corresponding shift in the run-in course line (time values) and a release range shift that is directly proportional to these factors.

In function HVSY, the **Dist. targ. to wind. edge** (row **seven**) is used to calculate the distance from the Target coordinates to the edge of the display window. It is used when determining display window size in the world coordinate (TTR range) system. Acceptable units for this value are meters (M), feet (F), or nautical miles (N).

The **Release Range** in row **eight** is the distance from the Target to the release point (0 seconds) along the course line. It assumes no wind correction factor and may be entered in meters (M), feet (F), or nautical miles (N).

Menu line **nine** contains the **Course Line** box, which is the desired flight heading in degrees (referenced to North). It is the angle used to draw the run-in line (x, y). Also, it is the azimuth line used when calculating all ranges (times) from the Target point. In the HVSY function, the grid is rotated so that it would reside on this line.

Test ID	Test Id		
Operation no	R123456		
Test Date	06/07/93		
Target name	HTGT		
Target X	254.2	Target Y	-1097.6
		Target Alt	1626.2 units M
Wind X	0	Wind Y	0 units M
Dist. targ. to wind edge	457.2	units	M
Release range	597	units	M
Course line (deg)	347		
Ground speed knots	551		
Normal alt. (AGL)	51	units	M
High alt. bound	152.4	units	M
Low alt. bound	30.48	units	M
Window max. alt.	304.8	units	M
Alt. tick interval	30.4	units	M
Target radius	114.3	units	M
Author's name	Ken Bauhs		
GENCAL segment file name	Gseg.asc	segment name	GSEG
HVSY segment file name	Hseg.asc	segment name	HSEG
Comments	This is comment line 1		
Comments			

Figure 2.2.10.1. TCALL - Segment Maker Editor

2.0 Operational Modes

Row **ten** is the **Ground speed** entry block. Its units must be in knots and it is used to compute time tics from the release point (0 seconds) to -360 seconds. This is the aircraft's Ground Speed (average) along the Course Line (Azimuth).

Menu lines **eleven through fifteen** are used by the HVSY function to define the **Altitude** grid lines. Each height value is referenced (offset from) the **Target H(msl)** value of line five. Each altitude value may be entered in units of meters (M), feet (F), or nautical miles (N). While values are entered in any of the three above units, they are converted to meters for all internal calculations (TCALL).

The **Normal alt.** (AGL) box must contain the Nominal Release (REL) altitude above the Target.

Block **High alt. bound** is used to enter the maximum (+) allowable release altitude above the Target.

Likewise, the **Low alt. bound** box is used to declare the minimum (-) release altitude above the Target.

The **Window max. alt.** block defines the maximum window height (top grid line) above the Target. Likewise, the **Target H(msl)** defines the minimum window height (bottom grid line).

Between these two height extremes (top and bottom) the **Alt. tick interval** is used to declare the distance between altitude grid lines (and the number of grid lines).

Row **sixteen** is the **Target radius** entry block and it can be entered as meters (M), feet (F), or nautical miles (N). For the GENCAL function, it is the radius of a circle centered at the Target location. In the HVSY function, it is the distance between tick marks around the Target tick mark.

The **Author's name** is an optional entry in **row seventeen**. It may be up to 19 characters long and is saved in the header blocks of both files (*.asc).

Menu line **eighteen** contains the **GENCAL segment file name** and its associated **segment name**. This is the run-in course line (x and y) ASCII Segment file (*.asc) output by the GENCAL function.

Row **nineteen** is used to enter the **HVSY segment file name** and its **segment name**. This is the altitude versus time segment file (*.asc) output by the HVSY function.

Both files are stored in directory '/users/egs/data' and have default extensions '.asc' for ASCII files. If the output files do not exist, they are created. If they already exist, they are overwritten with new data.

Note: No overwrite prompt is displayed. The filenames may be 15 characters long (maximum).

The Segment names are only used if the segment file is loaded as a main segment. Then it would be the default switch label for the PC Control Panel. The Segment Name is stored in each segment's file header and is a maximum of four (4) text characters long. It may be blank (empty).

Figures 2.2.10.3 and 2.2.10.4 show examples of both ASCII Segment Output files as displayed by the SHOW_SEG client program.

User comments may be entered in rows **twenty and twenty-one** in the blocks labeled **Comments**. A maximum of 40 text characters may be entered per line. These two lines are concatenated and subsequently stored inside the files, with the header data.

2.2.10.2 TCALL Functions

The first three push buttons, on the lower-left of the TCALL menu, are the screen Edit buttons. The values saved in this file "TCADEF" are the parameters appearing on the screen menu with the exception of the Test ID, Operation no., and Test Date blocks. These three blocks are saved in Command shared memory and the START default file "STADEF". The other screen values can be cleared (**New**), loaded from disk (**Load**), and saved to disk (**Save**). **Note:** The **Save** push button performs a complete save, including the first three parameters Test ID, Operation no., and Test Date, to "STADEF".

These screen menu files have little to do with the Graphics Real-Time Program (GRTP). The useful output from this client is stored in the two Segment Output files (*.asc). Screen menu files (i.e., "TCADEF") may be used as temporary scratch files, to save a menu setup, while performing another task. Also, this file's contents determine the default menu values when the client is started.

The Print push button lists the current contents of the TCALL menu to the line printer.

TCALL is a client specific Execute push button. Selecting this button causes the distance (Range) versus time calculation to be performed for specific time values. These time calculation values are stored in memory arrays for use by the GENCAL and HVSY functions. This is a fairly simple calculation as follows:

$$\text{Distance}(t) = \text{Ground Speed(knots)} * \text{Time} * 1852/3600$$

Where the Distance is displacement from the Release Point (0 seconds) along the Course Line Path (Heading). The Range from the Target along the flight path is then:

$$\text{Range}(t) = \text{Release Range} + \text{Distance}(t)$$

An example listing from the TCALL program is shown in Figure 2.2.10.2 for time values from 0 seconds (release time) to -360 seconds.

The **GENCAL** push button Executes a client function that takes the memory array values (TCALL) and constructs the Course Line Segment Output file (*.asc). An example of this output file is shown in Figure 2.2.10.3.

Note that function TCALL must have been run (at least once) before GENCAL can be executed. If GENCAL detects that TCALL has not been run, it will call and execute it (silently) for you. However, if TCALL has been run at least once and any of the menu screen values are changed the user must select the TCALL button to calculate the new values and store them in memory arrays.

HVSY is another client specific Execute push button. Selecting this button takes the memory array values (TCALL) and constructs the Height versus Distance (Time) Segment Output file (*.asc). An example of this output file is shown in Figure 2.2.10.4.

Again, function TCALL must have been run (at least once) before HVSY can be executed. If HVSY detects that TCALL has not been run, it will call and execute it (silently) for you.

Output From Program TCALL

Current date & time: Fri Feb 26 07:50:24 1993

Segment file name: Gseg.asc

Operation #: R123456 Date: 02/23/93 Target name : HTGT

Heading - 347.0 degrees
 Ground speed - 551.0 knots
 Release range - 597.0 meters - 1958.7 feet

Target coords X = 254.2 meters - 834.0 feet
 Y = -1097.6 meters - -3601.0 feet
 H = 1626.2 meters - 5335.3 feet

Wind correction X = 0.0 meters - 0.0 feet
 Y = 0.0 meters - 0.0 feet

Nominal altitude - 61.0 meters - 200.1 feet
 Max. altitude - 152.4 meters - 500.0 feet
 Min altitude - 30.5 meters - 100.0 feet
 Target radius - 114.3 meters - 375.0 feet

Time	METERS		X	Y	NMILES	FEET	
	Range					Range	
0	597		388	-1679	0.3	1959	1275
-1	880		452	-1955	0.5	2889	1484
-2	1164		516	-2232	0.6	3819	1693
-3	1447		580	-2508	0.8	4749	1902
-4	1731		644	-2784	0.9	5679	2111
-5	2014		707	-3060	1.1	6609	2321
-6	2298		771	-3336	1.2	7539	2530
-7	2581		835	-3613	1.4	8468	2739
-8	2865		899	-3889	1.5	9398	2948
-9	3148		962	-4165	1.7	10328	3157
-10	3432		1026	-4441	1.9	11258	3367
-15	4849		1345	-5822	2.6	15908	4413
-20	6266		1664	-7203	3.4	20558	5459
-25	7683		1983	-8584	4.1	25208	6505
-30	9101		2301	-9965	4.9	29858	7551
-35	10518		2620	-11346	5.7	34508	8597
-40	11935		2939	-12727	6.4	39158	9643
-45	13353		3258	-14108	7.2	43808	10689
-50	14770		3577	-15489	8.0	48457	11735
-55	16187		3896	-16870	8.7	53107	12781
-60	17604		4214	-18251	9.5	57757	13827
-90	26108		6127	-26537	14.1	85656	20102
-120	34612		8040	-34822	18.7	113556	26378
-150	43115		9953	-43108	23.3	141455	32654
-180	51619		11866	-51394	27.9	169354	38930
-210	60123		13779	-59680	32.5	197253	45206
-240	68627		15692	-67965	37.1	225153	51482
-270	77130		17605	-76251	41.6	253052	57758
-300	85634		19518	-84537	46.2	280951	64034
-330	94138		21431	-92822	50.8	308850	70310
-360	102641		23343	-101108	55.4	336750	76586

* * * * * END OF DATA * * * * *

Figure 2.2.10.2. Example Printer Output from TCALL

Figure 2.2.10.3. GENCAL Segment File - Course Line

Figure 2.2.10.4. HVSY Segment File - Altitude vs. Time

The **Quit** button halts this client program's execution, cleans up memory references, and removes the process from the process list. It is permanent in this session with the X Server. If the user is not sure whether this client may be needed again, minimize (icon) the TCALL window.

Button **Help Me!** causes a pop-up window to appear with any available help information.

2.0 Operational Modes

3.0 UTILITY PROGRAMS

3.1 Map (View) Conversion

The basic function of the EGS Graphics Real-Time (GRTP) program is to display a TTR background range map and plot radar tracks on this map. Hence converting existing Tektronix 4081 maps and creating new map segments/views was a very important EGS process. To accomplish this task the Starbase/Display List hierarchy files (*.vfl) files must be created (Appendix E).

Inside the integrated EGS Setup (X11) environment (Section 2.2.8) this Segment/View editing is performed by the **SHOW_SEG** and **BUILD** client programs. This is still the preferred (primary) method of performing the editing functions and Appendices G and H describe detailed steps for performing this task.

However, this section will describe another method, outside the X-Window system, which may be useful from time to time. Both methods produce the same results.

3.1.1 Line-by-Line Editors

Two other utility programs, **ConvMap** and **ViewBld**, were created to perform the same Segment/View editing task but outside X11 window environment. They are similar to the X11 Setup programs (**SHOW_SEG** & **BUILD**) and Figure 3.1.1 shows the development path required when utilizing these programs. One significant difference between the two methods is that the interactive (mouse) Window function is not available outside the X-server.

The first step is to create Segment text file(s) (*.com or *.asc) that serve as the input file(s) for the process. One method is to use the standard UNIX 'vi' editor to create them. Then the two programs **ConvMap** and **ViewBld** can convert them to the Starbase/Display List hierarchy form.

Note: Both programs **ConvMap** and **ViewBld** reside in directory '/users/egs/data/'.

WARNING: Neither **ConvMap** nor **ViewBld** can be executed inside the X-Window server environment because they grab control of the CRT display screen for drawing. This conflicts with the X server operation.

3.1.1.1 Segment Converter (ConvMap)

The program **ConvMap** is written in the 'C' language and uses Starbase/Display List graphical commands to display a Segment (*.sfl) file on the CRT screen. It accepts Command (*.com), ASCII text (*.asc), or integer Segment (*.sfl) files as input. If a Command (*.com) file is the input source, both ASCII text (*.asc) and Segment (*.sfl) files may (optionally) be output.

WARNING: Shutdown (exit) the X-Server.

To execute the **ConvMap** editor program, type the following at the UNIX prompt:

Figure 3.1.1. Building EGS Segments and Views


```
$ cd /users/egs/data <CR>
$ ConvMap input_file_name.asc <CR>
```

Optionally, the user can leave off the 'input_file_name' parameter and the program will prompt for it as follows:

Enter Input file name - input_file_name.asc <CR>

Note: If no extension is entered, a Command file (*.com) input is assumed.

Then, depending on the input extension entered (*.com, *.asc, or *.sfl), the **ConvMap** program will display none, one, or both of the following prompts:

Enter Starbase 'ascii' filename -
[CR - None], ['- ' - Infile.asc] -

Enter Display List 'segment' filename -
[CR - None], ['- ' - Infile.sfl] -

At this point, the user has three choices:

1. Enter the **filename** (*.asc or *.sfl) for the Output file.
2. Type '**Return**' <CR> if no Output file is desired.
3. Type '**-**' (Minus/Dash) if the Output file is to use the same name as the Input file but change the extension (*.asc or *.sfl).

If the designated Output file already exists, the following prompt message will appear:

'out_file_name' already exists, OK to overwrite ? -

Answering anything but [**Y**]es or [**y**]es aborts the Output file write operation.

Note: If the Input file is a Command file (*.com), the user will be prompted for additional information.

3.1.1.2 View Builder (ViewBld)

The program **ViewBld** is written in the 'C' language and uses Starbase/Display List graphical commands to construct/display a View (*.vfl) file. It accepts ASCII text (*.asc) or integer Segment (*.sfl) files as input. It produces a complete integer View (*.vfl) file for output, which contains the main segment and copies of all the subsegments which are called.

Optionally, **ViewBld** will produce an incomplete ASCII text (*.asc) output file. It is not complete due to the fact that it is an ASCII text listing of the main segment only. It may be used to reconstruct the View (*.vfl) file or in troubleshooting. Warning: It will cause an error if input to the ConvMap program.

WARNING: Shutdown (exit) the X-Server.

To execute the ViewBld program, type the following at the UNIX prompt:

3.0 Utility Programs

```
$ cd /users/egs/data <CR>
$ ViewBld out_file_name.asc <CR>
```

Optionally, the user can leave off the 'out_file_name' parameter and the program will prompt for it as follows:

```
Enter Output 'ascii' file name - out_file_name.asc <CR>
```

The user may enter one of two choices:

1. Enter the **filename** for the ASCII text Output file (*.asc).
2. Type **'Return'** <CR> if no ASCII Output file is desired.

Note: If no extension is entered, a ASCII file (*.asc) Output file is assumed.

Next the **ViewBld** program will now display the following prompt:

```
Enter Output 'View' filename [CR - None],['-' - Ascfile.vfl] - - <CR>
```

At this point, the user has three choices:

1. Enter the **filename** for the 'View' Output file (*.vfl).
2. Type **'Return'** <CR> if no 'View' Output file is desired.
3. Type **'-'** (Minus/Dash) if the 'View' Output file is to use the same name as the ASCII file but change the extension (*.vfl). An ASCII file must have been declared.

In either of the above cases, if the named Output files (*.asc or *.vfl) already exist the following prompt message(s) will appear:

```
'out_file_name' already exists, OK to overwrite ? - Y <CR>
```

Answering anything but [Y]es or [y]es aborts that particular Output file operation.

The user will now be prompted for additional setup information as follows:

```
Enter Switch (View)# [1 <-> 8] - 1 <CR>
Enter 'Viewname' [4 chars.-Max.] - name <CR>
```

Type in a number, between 1 and 8, for the Switch Number.

Type in a 'View' label for this View Output file. It is the Switch Label that will appear on the PC Control Panel's View select buttons.

In the Starbase CMD window, a prompt will appear asking for the Author's name:

```
Enter Author - Unknown <CR>
```

Type in the person's name who is constructing this 'View' file.

Next the View Window's default viewport (world coordinates) are requested:

[R -> Range Defaults; S -> Section Defaults; QUIT -> Ends Program]

Enter: View Window - lower/left corner <x0, y0> (meters) - x,y <CR>

The program expects one of the following:

1. Enter TTR range **coordinate pair** <x0,y0> defining the lower/left corner of the 'View' window in meters.
2. Enter 'R' (or 'r') for a Range map coordinate viewport.
 <x0, y0> = -16510.0, -21590.0 meters
 <x1, y1> = 21590.0, 16510.0 meters
3. Enter 'S' (or 's') for a Section map coordinate viewport.
 <x0, y0> = -259000.0, -315000.0 meters
 <x1, y1> = 301000.0, 245000.0 meters
4. Typing 'QUIT' (or 'quit') stops the program.

If option #1 was selected, the following prompt will be displayed:

[QUIT -> Ends Program]

Enter: View Window - upper/right corner <x1, y1> (meters) - x,y <CR>

The program expects one of the following:

1. Enter TTR range **coordinate pair** <x1,y1> defining the upper/right corner of the 'View' window in meters.
2. Typing 'QUIT' (or 'quit') stops the program.

Finally, the following prompt will be displayed recursively, up to eight (8) times:

Enter Segment# [?] filename [CR->Last Segment, QUIT] - in_name <CR>

In response, the program expects one of the following:

1. Enter the **filename** of the Input Segment file (*.asc or *.sfl) to be used for this View Segment# [?].
2. Type 'Return' <CR> if done entering Segments.
3. Typing 'QUIT' (or 'quit') stops the program.

Note: The Input filename extension must be provided or an error occurs.

After each Segment filename is entered it is displayed in the graphics window.

If the 'QUIT' option is selected, the work up to that point is saved in the respective files (*.asc and/or *.vfl).

3.1.2 CAD Graphical Editor (me10)

At the present time, an HP Mechanical Engineering (Computer Aided Design) package has been installed on the HP370 system. It allows the user to create drawings to scale in meters, feet, or miles.

The output of this package is an industry standard ASCII file (Model Interface - MI). Unfortunately this MI file is not directly compatible with the EGS graphics Starbase/Display List format. Current enhancements are to write a graphics file translator and integrate this package into the EGS graphics system.

3.2 Diagnostics

During the process of developing the EGS software, several diagnostic programs were written. These programs were used to both debug the system and verify proper operation. They are still extremely useful when troubleshooting any future problems (hardware and/or software) that may arise.

3.2.1 Simulator (Target_sim)

On the HP370, a Target simulator program may be used to simulate input data from the uVax computer over the parallel gpio interface. The program runs on the local HP370 computer and writes data to the Target Shared Memory Block ("Target_shm.h"). This Target Data Block is described in Appendix B.

The Target simulator "Target_sim" program is completely independent of the Graphics Real-Time Process (GRTP) and runs asynchronously in parallel with it. It must be executed from a separate terminal (RS-232) connected to the HP370. Alternatively it may be executed over a LAN network with a terminal emulator ("rlogin") program. Figure 3.2.1.1 shows how the simulator (Target_sim) works with the rest of the EGS system.

An important point to recognize is that the Graphics Real-Time Process (GRTP) runs exactly as it would if data were being received from the uVax computer. In fact the daemon I/O process "pollgpio" can also be active at the same time as the simulator. However, the uVax parallel data output should be disabled or some very strange results will occur. This is because both processes, "Target_sim" and "pollgpio", are writing to Target Shared Memory at the same time with different data.

The "Target_sim" program is written in the 'C' language and utilizes Starbase calls for event handling (Softkeys). Therefore, the terminal must be a type which is recognized by the Starbase library (just about any HP terminal).

Every value in the Target Shared Memory Data Block (Appendix B) is generated by the simulator. Certain selected parameters may be changed from the terminal keyboard by using the **User Function keys** [F1 <-> F8].

The simulator program generates tracking vector data for all (TOTAL_TARGETS macro) Radars in the system. The vector pattern generated is a 'Circle-of-circles' as shown in Figure 3.2.1.2. The center of this pattern is TTR range zero (0,0).

Note: To run either "Target_sim" or "Egs_master" the user must be in directory '/users/egs/graphics'.

Figure 3.2.1.1. HP370 Target Simulation ("Target_sim") and Intercommunication

3.0 Utility Programs

To start the Target simulator, type the following in response to the UNIX prompt on the auxiliary terminal:

\$ Target_sim <CR>

Start the Graphics Real-Time Program as you normally would. If it is not already executing, type the following at the System Console:

\$ Egs_master <CR>

Once the simulator "Target_sim" is started, Function keys [F1 <-> F8] will be active.

Radar object tracking speed (motion) is determined by the combination of the circle radius (R) distance and the Angle Update Increment (θ) (function key F1). Actually, the track point (x,y) moves a distance along the circumference of each radar's circle that is directly proportional to this angle θ . Its default value is 1 degree per loop interval (.1 second).

Function key F2 (Radar Spacing) allows the angle between each Radar's circle (Ω) to be changed. By default it is set to 45 degrees at program startup.

The radius (R) of each Radar's trajectory (circle) is 2000 meters (default) when the simulator starts up. It may be changed by depressing the F3 function key. Also, this value (R) determines the radius (2R) of the major circle pattern. The altitude values (z and h) are also derived from this parameter.

Function key F4 allows the course line (cl) value to be changed. By default it starts as 170 degrees.

Any Command String can be sent to the Command Decoder "Command_proc" by utilizing the F5 function key. The command strings are defined in the memo 'Control Panel Switch Functions, Enhanced Graphics System' by D. F. Anderson (August 9, 1991). Note only the required ASCII text string should be entered; the 'StxG' preamble is automatically appended.

Function keys F6 and F7 allow the Radar A and Radar B Indices in Target Shared Memory to be changed. This is slightly different than sending the 'RADAR SELECT A'(O1 <-> O8) or 'RADAR SELECT B' (P1 <-> P8) command strings (F5). At startup Radar A is set for Radar#1 (Rad0) and Radar B is set for Radar#2 (Rad1).

To stop the simulator program, press the F8 function key. Then answer [y]es to the following prompt:

Do you want to quit [Y or N] - y

Other simulator Target Shared memory data values are fixed. The fixed parameters are as follows:

Reference Point "REF0" - Main Lake Target

x = 0 meters
y = 0 meters
z = 1625 meters
h = 1625 meters

Operation Number = "Rsimulator"

Note: Update Rate: 10 times/sec. - Tracking Vectors
1 time/sec. - Strings

Note: All Units are in Meters.

Altitude: Radar# 1 = 0R meters
Radar# 2 = 1R meters
Radar# 3 = 2R meters
Radar# 4 = 3R meters
Radar# 5 = 4R meters

Function	F1 - Angle Increment(DTheta - θ) - degrees($1 < \theta < 360$)
Keys:	F2 - Radar Spacing (Omega - Ω) - degrees($0 < \Omega < 360$)
	F3 - Circle Radius (R) - meters
	F4 - Course Line (cl) - degrees($0 < cl < 360$)
	F5 - New Command (Cmd)
	F6 - Radar A Index (ai) - integer($0 < ai < 7$)
	F7 - Radar B Index (bi) - integer($0 < bi < 7$)
	F8 - Quit ? (c)

Figure 3.2.1.2. Target Simulator Radar Vector Pattern

3.0 Utility Programs

The Date String uses the current 'Date' as read from the HP370 UNIX operating system.

Both the 'Time-of-Day' and 'GMT Time' are derived from the UNIX system clock and the time since the simulator program was started is used.

3.2.2 Target Shared Memory Display (Target_dsp)

The diagnostic program "Target_dsp" allows the contents of the Target Shared Memory Block ("Target_shm.h") to be displayed in real time. This program provides a dynamic view of the Target Data Block and displays it as user-readable text. Appendix B describes this Data Block in greater detail.

The "Target_dsp" program replaces the normal graphics process "Graphics_proc" in the EGS Graphics Real-Time system. Otherwise, the remaining EGS modules ("Egs_master", "pollgpio", and "Command_proc") are the same as in the normal setup. Figure 3.2.2 shows how the "Target_dsp" program fits in with the rest of the EGS system. It is implemented using simple UNIX file renaming ('mv') and copy ('cp') commands.

Program "Target_dsp" is written in the 'C' language and utilizes both Starbase and Fast Alpha library calls. The Alpha text is displayed on the CRT screen in what is usually reserved as the **A Window** in the normal Graphics Process.

Note: The normal Command Decoder Process "Command_proc" is still active and Commands are accepted. However, commands implemented in the normal graphics program will not be carried out.

The display update rate for the Target Data values is nominally the currently selected "Peck Rate" as set at the PC Control Panel. The default display update rate is 1 time/second.

Note: To run the "Target_dsp" diagnostic utility the user must be in the directory '/users/egs/graphics'.

```
$ cd /users/egs/graphics <CR>
```

WARNING: Save the current Graphics Process "Graphics_proc" before it is replaced with the "Target_dsp" program or it will be lost. Type the following at the UNIX prompt:

```
$ cp Graphics_proc graphics <CR>
```

To replace the current version of the Graphics Process, type the following at the UNIX prompt:

```
$ cp Target_dsp Graphics_proc <CR>
```

Start the Graphics Real-Time Program (GRTP) as you normally would or type the following at the System Console:

```
$ Egs_master <CR>
```

Figure 3.2.2. Target Shared Memory (Target_dsp) and Intercommunication

3.0 Utility Programs

To halt the program's execution use the normal shutdown procedure from the PC Control Panel. Another method to stop the Graphics Program is to press the F8 function key. Then answer [y]es to the following prompt:

Do you want to quit [Y or N] - y <CR>

Once the Graphics Real-Time Program has halted, the normal Graphics Process must be reloaded as follows (for normal operation):

\$ mv graphics Graphics_proc <CR>

This removes any previous files with the "Graphics_proc" name and renames the "graphics" save file to "Graphics_proc".

3.2.3 Command Shared Memory Display (Commnd_dsp)

The diagnostic program "Commnd_dsp" allows the contents of the Command Shared Memory Block ("Command.h") to be displayed in real time. This program provides a dynamic view of the Command Data Block and displays it as user-readable text. Appendix C describes this Data Block in greater detail.

The "Commnd_dsp" program replaces the normal graphics process "Graphics_proc" in the EGS Graphics Real-time system. The remaining EGS modules ("Egs_master", "pollpio", and "Command_proc") are the same as in the normal setup. Figure 3.2.3 shows how the "Commnd_dsp" program fits in with the rest of the EGS system. It is implemented using simple UNIX file renaming ('mv') and copy ('cp') commands.

Program "Commnd_dsp" is written in the 'C' language and utilizes both Starbase and Fast Alpha library calls. The Alpha text is displayed on the CRT screen in what is usually reserved as the **A Window** in the normal Graphics Process.

Note: The normal Command Decoder Process "Command_proc" is still active and Commands are accepted. However, commands implemented in the normal graphics program will not be carried out.

The display update rate for the Command Data values is nominally the currently selected "Peck Rate" as set at the PC Control Panel. The default display update rate is 1 time/second.

Note: To run the "Commnd_dsp" diagnostic utility the user must be in the directory '/users/egs/graphics'.

\$ cd /users/egs/graphics <CR>

WARNING: Save the current Graphics Process "Graphics_proc" before it is replaced with the "Commnd_dsp" program or it will be lost. Type the following at the UNIX prompt:

\$ cp Graphics_proc graphics <CR>

To replace the current version of the Graphics Process, type the following at the UNIX prompt:

\$ cp Commnd_dsp Graphics_proc <CR>

Figure 3.2.3. Command Shared Memory (Commnd_dsp) and Intercommunication

3.0 Utility Programs

Start the Graphics Real-Time Program (GRTP) as you normally would or type the following at the System Console:

```
$ Egs_master <CR>
```

To halt the program's execution use the normal shutdown procedure from the PC Control Panel. Another method to stop the Graphics Program is to press the F8 function key. Then answer [y]es to the following prompt:

```
Do you want to quit [Y or N] - y <CR>
```

Once the Graphics Real-Time Program has halted, the normal Graphics Process must be reloaded as follows (for normal operation):

```
$ mv graphics Graphics_proc <CR>
```

This removes any previous files with the "Graphics_proc" name and renames the "graphics" save file to "Graphics_proc".

3.2.4 Print Display List (PrtDLst)

Another useful utility program is the Print Display List ("PrtDLst"), which displays the contents of integer Display List files in a user-readable form (text).

As mentioned previously, the map (view) files for the EGS Graphics system may be stored in any of four file formats (Appendix E). These four formats are:

1. ASCII File Format
2. Segment File Format
3. View File Format
4. Bitmap Image Format

In the normal setup procedure for a test, the operator creates an ASCII file (*.asc or *.com). Then the setup program "SHOW_SEG" (ConvMap) operates on this ASCII file to produce a Segment file. The Segment file is a binary (integer) file that is defined in the HP Display List library. The whole procedure is analogous to a 'compiled' version of the ASCII file, which executes very quickly.

The procedure for creating View files is almost identical, except the hierarchy is implemented. The setup program 'BUILD' (ViewBld) takes Segment (or ASCII) files and produces a View file. The View file is in the same binary (integer) format as the Segment file but expanded.

Program "PrtDLst" is written in the 'C' language and utilizes the Starbase/Display List library calls.

This binary (integer) format is not easily readable by the average person. Program "PrtDLst" performs an operation that is analogous to a 'reverse-compile' on the integer file. It produces a user-readable text output from a binary Segment or View Display List file.

Note: The text output is not exactly the same as an ASCII (*.asc) file because some information is lost in the conversion process.

Examples of both the ASCII file format (section I) and the integer file formats (section II) are included in Appendix E. The Display List examples in section IIE (Segment file) and section IIF (View file) were produced with this "PrtDLst" utility program.

Note: To run the "PrtDLst" utility the user must be in the directory '/users/egs/data'.

```
$ cd /users/egs/data <CR>
```

WARNING: Do not attempt to execute "PrtDLst" from within the integrated (X-Window) setup environment. It attaches to the Starbase library which grabs control of the CRT screen.

The normal mode of output is to display the listing on the Console screen. To run "PrtDLst" in this mode, type the following in response to the UNIX prompt:

```
$ PrtDLst file_name.sfl <CR>
```

To send the program output to a printer, type the following:

```
$ PrtDLst file_name.vfl | lp <CR>
```

To send the program output to a disk file, type the following:

```
$ PrtDLst file_name.sfl >file_name.txt <CR>
```

3.2.5 TTR Coordinates

While developing the HP370 Graphics Real-Time System a coordinate transformation library ("ttrcoord.c", "rngcoord.c" and molod.c) was created. The libraries are further described in Section 4.4. In order to verify the library routine's operation, several diagnostic utility programs were written. The results were then compared with tables that are currently used at Tonopah Test Range (TTR).

Each of these programs is written in the 'C' language and is transportable to other machines with a 'C' compiler. Therefore, a listing of all programs and their functions are included here for future reference.

The first set of programs were developed based on the North American Datum of 1927 (NAD27) and the transformation equations are shown in Appendix F.

Subsequent work expanded the library to include transformations for the WGS84 ellipsoid coordinate system.

PROGRAM NAME	FUNCTION
1. ttrxyz	Program to convert (NAD27) latitude, longitude, and height coordinates into TTR range (x,y,z) coordinates.
1a. ttrxyz84	Program to convert (WGS84) latitude, longitude, and height coordinates into TTR range (x,y,z) coordinates.
2. ttrllh	Program to convert TTR range (x,y,z) coordinates to (NAD27) latitude, longitude, and height coordinates.
2a. ttrllh84	Program to convert TTR range (x,y,z) coordinates to (WGS84) latitude, longitude, and height coordinates.
3. ttr_rae	Program to convert a TTR range (x,y,z) coordinate to Range, Azimuth, and Elevation (RAE) value. Relative to TTR Range zero and/or to a second set of range coordinates.
4. ttrmdcl	Program to convert a TTR range (x,y,z) coordinate to Range, Azimuth, and Elevation (RAE) value. Relative to TTR range zero and then find the magnetic heading from the Azimuth.

4.0 EGS LIBRARIES

An extensive set of library functions were developed to support the EGS Graphics Real-Time (GRTP) System. These functions are used in several different programs, from the Setup editors to the actual real-time plotting process. They provide a consistent way of handling the various functions across the many different operations.

At the present time three separate EGS libraries exist for the HP370 Graphics system. They are summarized as follows:

1. "util.a" - EGS Utility I/O Library.
2. "libdas.a" - X-Window Graphical User Interface (GUI) Library.
3. "egslib.a" - EGS Graphics Real-Time Library.

The EGS Utility I/O Library ("util.a") is an HP archive library that controls access to both the Target and Reply Shared Memory Blocks. Every program that uses data in these shared memory blocks must use the macros/functions in this library to attach/detach and read/write. This limits access to only qualified user programs and forces consistent behavior when accessing data. The routines in this library ("util.a") were written by W. D. Love (SNL/TTR - 2719). To include these library macros/functions in a program, the include file "egs_defs.h" must be declared in each source file.

The X-Window Setup Library ("libdas.a") is a collection of 'X Widget and Intrinsic' based functions which form a Graphical User Interface (GUI) for the integrated Setup environment. Contact K. C. Bauhs (SNL - 2742) for additional information on this library and its usage.

Library "egslib.a" contains all the macros and functions which form the basis of the Starbase/Display Graphics Interface. Routines for both the EGS Graphics Real-Time Program (GRTP) and the Segment/View Editors are included in this library. In addition, the TTR coordinate transformation functions are contained in this library. It is a large HP archive library that is further described in the subsequent sections. Due to its large size, it has been further divided into the following general classifications:

- 4.1 Segment File handling - Display List Integer file (sfile) editor functions and interpreter.
- 4.2 ASCII File handling - ASCII file handling editor and TTR Command file converter.
- 4.3 EGS Real-Time Functions - Starbase graphics functions and Forms utility functions.
- 4.4 TTR Coordinate Conversions

Each of these general topics is subdivided according to source modules. This was done in order to keep the logistics manageable and each source module to less than 1500 lines of code.

All source code in the "egslib.a" library is written in the 'C' language. It references the HP Starbase/Display List libraries and the HP Fast Alpha/Font Manager library.

The following sections contain a listing of each externally callable function, a description of what it does, and a description of how to use it.

For convenience a Master Index of all functions is also given.

Master Index (egslib.a)

Segment File Handling (sfile.h)

AddSection	- add Section to the 'Display_List'.
DeleteSection	- delete a Section of the 'Display_List'
evl_exc_type	- evaluate/execute a 'Starbase/Display_List'
get_ele_type	- get element type (EGS) from Command string.
get_view_data	- reads and converts AppData I structure.
newline	- gets another line(s) from Command String.
parse_str	- parse Command string/return (form) values.
PrintSegment	- print 'ASCII' representation of a segment.
read_display_list	- reads and converts an "sfile" format file.
read_display_list_segment	- reads and converts an "sfile" segment.
RefreshSection	- redraw a Section of the 'Display_List'.
ReplaceSection	- replace Section of 'Display_List'.
set_seg_parms	- renames segments and filter namesets.
write_display_list	- writes a 'Display_List' in "sfile" format.

ASCII File Handling (sfile.h)

Asc_copy	- copies a Starbase/Display List 'ascii' buffer.
com_to_asc	- translates TTR commands to Starbase commands.
form_DataI	- fill an Application Data I structure.
form_DataII	- fill an Application Data II structure.
form_Hdr	- formats a Starbase/Display 'ascii' header.
get_Input_Buf	- reads a buffer of text from Input file.
open_files	- gets filenames, checks validity, and opens files.
read_asc_in	- read a buffer of text from Input file.
write_asc_out	- write a buffer of text to Output file.

EGS Real-Time Functions (egslib.h)

add_hack_lst	- add 'hack' element at end of linked list.
add_label_lst	- add 'label' element at end of linked list.
add_line_lst	- adds 'line' element at end of linked list.
Checkstring	- compares two strings (ignoring case).
choice_string	- returns a string from the keyboard.
Clear_AER	- clears AER window.
Clear_Hacks	- erases all Hacks from screen (A and B windows).
Clear_Labels	- erases all screen labels (A & B windows).
Clear_REF	- clears REF window.
close_graph_devices	- closes all the graphics devices.
Concat2D	- performs a matrix multiply of 3x2 matrices.
Display_AER	- displays AER pop-up window.
Display_REF	- display reference point - REF pop-up window.
Draw_Line	- draws a line in the A window.

ESG Real-Time Functions (egslib.h) (continued)

Form_32	- forms (3x2) matrix from (3x3) matrix.
Form_33	- forms (3x3) matrix from (3x2) matrix.
free_hack_list	- frees all memory from 'hack' linked list.
free_label_list	- frees all memory from 'label' linked list.
free_line_list	- frees all memory from 'line' linked list.
get_key_command	- gets command from 'keyboard' input device.
Get_Refln	- gets New Reference Point from keyboard.
Identity2D	- forms 2-dimensional Identity Matrix (3x2).
init_Font_Manager	- initializes Font Manager fonts.
init_graph_devices	- initializes all the graphics devices.
init_wndow	- initializes a graphics window parameter.
parse_lat_lng	- parses Latitude and Longitude Strings.
Plot_Radar_Sym	- plots a Radar Symbol (A and B windows).
Plot_Symbol	- plots a symbol in A and B windows.
ProcessKey	- processes a keyboard value.
Remove_Com_Shm	- removes 'Communication' Shared Memory Block.
restore_popup	- restores current background raster memory.
Rotate2D	- forms 2-dimensional Rotation Matrix (3x2).
save_popup	- saves current background raster memory.
Scale2D	- forms 2-dimensional Scaling Matrix (3x2).
SetCom_defaults	- set default values in 'Comm' Shared Memory.
Set_CTM	- sets Current Transformation Matrix values.
Set_Softkeys	- sets up the User Softkeys (Function keys).
Translate2D	- forms 2-dimensional Translation Matrix (3x2).
UpDate_Hacks	- updates Hacks (A and B windows).
UpDate_Labels	- updates labels (A and B windows).
UpDate_Trail	- updates the screen trail (A and B windows).
Write_Hack	- writes a Hack symbol (A and B windows).
Write_Label	- writes label (time and altitude) (A and B window).

FORMS Library (egslib.h)

form3	- Alpha Numeric Form#3 Template (Air Drop).
form3_fill	- Alpha Numeric Fill#3 Text (Air Drop).
form4	- Alpha Numeric Form#4 Template (Blank).
form4_fill	- Alpha Numeric Fill#4 Text (Blank).

TTR Coordinate Conversions (ttrcoord.h)

conv_magn	- adjusts Azimuth (True North to Magnetic North).
conv_truen	- adjusts Azimuth (Magnetic North to True North).
deg_sdms	- converts double to "Degrees Minutes Seconds".
llh	- Earth Centered (NAD27) -> geodetic (iterative).
llh72	- Earth Centered (WGS72) -> geodetic (iterative).
molod	- Molodensky datum transformation (WGS84 -> NAD27).
rae_xyz	- Range, Azimuth, and Elevation to range (x,y,z).
rotat1	- TTR coordinates -> parallel to Earth Centered.
rotat2	- parallel to Earth Centered -> TTR coordinates.

TTR Coordinate Conversions (ttrcoord.h) (continued)	
rotate_cl	- translate/rotate coord. to ref. and Course Line.
sdms_deg	- converts "Degrees Minutes'Seconds" to double.
tearth_rng	- translation Earth Centered (NAD27) -> TTR.
tearth_rng72	- translation Earth Centered (WGS72) -> TTR.
time_to_go	- calculates time remaining to ref. (at velocity).
trng_earth	- translation TTR -> Earth Centered (NAD27).
trng_earth72	- translation TTR -> Earth Centered (WGS72).
wgsnad	- converts WGS84 coordinates to NAD27 coordinates.
xyz	- geodetic -> Earth Centered (NAD27).
xyz72	- geodetic -> Earth Centered (WGS72).
xyz_rae	- range (x,y,z) to Range, Azimuth, and Elevation.

4.1 Segment File (sfile.h)

The Segment file library contains functions which facilitate the handling of the HP Starbase/Display List binary (integer) graphics files. This includes both the Segment and View file formats described in Appendix E. It is a comprehensive set of **editing** functions which allows Segment files to be changed on-line. In addition, routines are provided to read and write **disk I/O** Segment files. Finally, the line-by-line ASCII to Display List **interpreter** (converter) is included with this library. The constants and prototypes for this library are in "sfile.h" (Appendix J).

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

AddSection

SOURCE

"sfileb1.c"

PURPOSE

Add a Section to the 'Display_List' starting at Label, End of Segment, or Current pointer.

SYNOPSIS

```
#include "sfile.h"

int AddSection(fildes, Label, SecArray, Sec_size) ;
    int  fildes ;                /* Starbase file descriptor */
    int  Label ;                 /* Label from which to start
                                appending */
    char SecArray[] [LINE_LENGTH]; /* text array which holds
                                New_Section info. */
    int  Sec_size ;              /* size of section array
                                (lines) */
```

DESCRIPTION

This routine allows a 'Display_List' section to be added to an existing 'Display_List' starting at **Label** or ...

if **Label == NO_LABEL** start from current pointer.
if **Label == END_LABEL** append at end of 'Display_List'.

Returns number of lines actually appended.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

RefreshSection, DeleteSection, ReplaceSection

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

DeleteSection

SOURCE

"sfileb1.c"

PURPOSE

Delete a Section of the 'Display_List' from a Label to another label (or end of Segment).

SYNOPSIS

```
#include "sfile.h"

void DeleteSection(fildes, Label) ;
    int    fildes ;        /* Starbase file descriptor */
    int    Label ;         /* Label from which to start delete */
```

DESCRIPTION

This routine allows a 'Display_List' section to be deleted from an existing 'Display_List' starting at Label and stopping at next label or end of Segment if no following label exists.

if Label == NO_LABEL start from current pointer.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

RefreshSection, AddSection, ReplaceSection

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

evl_exc_type

SOURCE

"sfilelb2.c"

PURPOSE

Evaluate and execute a 'Starbase/Display_List' command from an 'ASCII' Command string format.

SYNOPSIS

```
#include "sfile.h"

int evl_exc_type(type, fildes, Strng, Sec_size) ;
    int    type ;           /* element type - from 'sbd1.c.h' */
    int    fildes ;         /* Starbase file descriptor */
    char   *Strng ;         /* 'ASCII' Command string */
    int    Sec_size ;       /* Command string array size (lines) */
```

DESCRIPTION

This routine executes a 'Starbase/Display_List' command from an 'ASCII' string format and places the result in the currently active 'Display List'(fildes) memory.

The integer element type is as defined in the 'Starbase' include file 'sbd1.c.h' and may be obtained by calling library function 'get_ele_type'. This function uses the 'type' passed in the parameter list not in the string 'Strng'.

The 'Starbase' device file descriptor 'fildes' is used in place of any value which may be present in the command string 'Strng'.

Parameter 'Sec_size' is the number of additional lines in the Command string (if needed).

Returns number of lines used or zero (0) on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

get_ele_type

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

get_ele_type

SOURCE

"sfileb1.c"

PURPOSE

Get EGS element type from 'Starbase/Display_List' array string format.

SYNOPSIS

```
#include "sfile.h"
int get_ele_type(Strng) ;
    char *Strng ;          /* 'ASCII' Command string */
```

DESCRIPTION

This routine gets a 'Starbase/Display_List' element type from an 'ASCII' string. If it is not a supported Starbase command in the mappable EGS Starbase/Display List Command set a 'NOP' type is returned.

The returned integer element type is as defined in the Starbase include file 'sddl.c.h'.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

evl_exe_type

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

get_view_data

SOURCE

"sfilebl1.c"

PURPOSE

Reads an ASCII or integer Segment disk file, looks for the Application Data I element, and returns header information.

SYNOPSIS

```
#include "sfile.h"

int get_view_data(Pathname, view) ;
    char    Pathname[] ;      /* Path/name of file to open */
    map_ptr view ;            /* Pointer to Map/View Structure*/
```

DESCRIPTION

This routine gets the contents of a 'Starbase/Display List' Application Data I (header) element from a disk file. The file may be an 'ASCII' (*.asc) or an integer Segment (*.sfl / *.vfl) type. The 'map_ptr' structure is filled on success and is defined in the 'Command.h' include file (Appendix C).

```
typedef struct {
    int    swt;                /* control panel switch# */
    char    view_name[5];      /* view label name */
    char    path[22];          /* path for filename */
    char    filename[16];      /* filename for map */
    float    xoffset;          /* map zero offset from x */
    float    yoffset;          /* " " " y */
    float    zoffset;          /* " " " z */
    float    zoom ;            /* map magnification factor*/
    int    orientation;        /* map orientation - north */
    float    matrix[3][2];     /* transformation matrix */
} map_block, *map_ptr;
```

The returned value is zero (0) on success or negative on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Command.h

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

newline

SOURCE

"sfileb1.c"

PURPOSE

Gets another line(s) from the Command String array - skips blank lines (if required).

SYNOPSIS

```
#include "sfile.h"

int newline(Strng, Sec_size, sp, slen, lines) ;
    char *Strng ;    /* Address for Command String Array */
    int Sec_size ;   /* Number of additional lines in String
                      Array */
    int *sp ;        /* current string pointer location
                      Zeroed) */
    int *slen ;      /* current string length (reset to
                      new length) */
    int *lines ;     /* current line counter (updated) */
```

DESCRIPTION

This routine gets a new line from the String Input Buffer array and updates the string parameter counters for the line-by-line interpreter. Blank lines are skipped if they are encountered. It uses recursive calls until a non-blank line is found or the end of the String array is encountered.

Returns TRUE(1) if a new string is found or FALSE(0) if a new string line could not be found.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

parse_str, evl_exe_type

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

parse_str

SOURCE

"sfilebl.c"

PURPOSE

Parses the Command String 'Strng' and returns requested values in the desired format.

SYNOPSIS

```
#include "sfile.h"
int parse_str(form, num, arr, Strng, sp, slen, lines, Sec_size);
    char form ;      /* format conversion to perform */
    int num ;        /* number of conversions to perform */
    void *arr ;      /* pointer to return array (form) values*/
    char *Strng ;    /* Command String to be parsed (updated) */
    int *sp ;        /*current string pointer location(updated)*/
    int *slen ;      /* current string length (updated) */
    int *lines ;     /* number of lines used in array (updated)*/
    int Sec_size ;   /* Command string array size (lines) */
```

DESCRIPTION

This routine parses a Command String for a parameter value(s) and performs the requested format (form) conversion before returning it. Multiple values may be requested by a single call but they must be the same format type (form).

Calls function 'newline' if an end-of-line is encountered.

The following format (form) conversions are recognized.

- 'i' -> integer conversion
- 'f' -> floating point conversion
- 'c' -> string copy
- 'p' -> pointer conversion (nothing)

Note: The return type of '*arr' must agree with the format.

Returns number of conversions completed or zero (0) on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

newline, evl_exe_type

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

PrintSegment

SOURCE

"sfilebl.c"

PURPOSE

Prints an 'ASCII' representation of a Starbase Segment file descriptor.

SYNOPSIS

```
#include "sfile.h"

void PrintSegment(fildes, SegmentNo)
    int  fildes ;      /* Starbase file descriptor */
    int  SegmentNo ;   /* Segment number to print */
```

DESCRIPTION

This routine lists to the console screen a user-readable 'ASCII' text output of each element in a Starbase/Display List Segment. The complete Segment is listed.

Note: The desired Segment must currently be loaded into memory and the Starbase file descriptor must be initialized.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

read_display_list

SOURCE

"sfilebl.c"

PURPOSE

Reads and Converts a Starbase Segment ("sfile") format disk file into 'Display_List' memory.

SYNOPSIS

```
#include "sfile.h"

int read_display_list(fildes, Pathname) ;
    int fildes ;      /* Starbase file descriptor */
    char *PathName ; /* Path/Name of input Segment file */
```

DESCRIPTION

This routine opens and reads a Segment (integer) file into Starbase/Display List memory. It reads and interprets each Segment of the file by calling the function 'read_display_list_segment'.

Note: The Starbase file descriptor must previously be initialized.

Returns zero (0) on success or negative on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

write_display_list, read_display_list_segment

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

read_display_list_segment

SOURCE

"sfilebl1.c"

PURPOSE

Reads and Converts a Starbase Segment from a disk file (integer) into 'Display_List' memory.

SYNOPSIS

```
#include "sfile.h"

int read_display_list_segment(fildes, fp) ;
    int    fildes ;           /* Starbase file descriptor */
    FILE   *fp ;              /* Input file pointer */
```

DESCRIPTION

This routine reads a Segment (integer) from a file into Starbase/Display List memory. It reads and interprets each element of the file segment by calling the Starbase function 'interpret_ele'.

The input file (*fp) must already be opened and positioned to the start of the segment.

Note: The Starbase file descriptor must previously be initialized.

Returns zero (0) on success or negative on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

read_display_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

RefreshSection

SOURCE

"sfilebl.c"

PURPOSE

Redraw a Section of the 'Display_List' from a Label to the next label or the end of the segment.

SYNOPSIS

```
#include "sfile.h"

void RefreshSection(fildes, Label) ;
    int    fildes ; /* Starbase file descriptor */
    int    Label ;  /* Label from which to start refresh */
```

DESCRIPTION

This routine allows a 'Display_List' section to be redrawn on the Starbase device starting at **Label**. Only a portion of the Starbase Display List need be redrawn.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

AddSection, DeleteSection, ReplaceSection

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

ReplaceSection

SOURCE

"sfilelb1.c"

PURPOSE

Replace a Section of the 'Display_List' starting at Label or append to End of Segment.

SYNOPSIS

```
#include "sfile.h"

int ReplaceSection(fildes, Label, SecArray, Sec_size) ;
    int  fildes ;                /* Starbase file descriptor */
    int  Label ;                 /* Label from which to start replace
                                or append */
    char SecArray[ ][LINE_LENGTH]; /* text array which holds
                                New_Section info. */
    int  Sec_size ;              /* size of section array (lines) */
```

DESCRIPTION

This routine allows a 'Display_List' section to be replaced or added to an existing 'Display_List' starting at Label or ...

if Label = LABEL_UND append at end of Segment.

Note: If size of New Section is less than the size of Old Section it is replacing, delete the extra lines. Likewise if the size of the New Section is larger than the Old Section, insert (append) the extra lines at the replacement location.

Returns number of lines actually replaced/appended.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

RefreshSection, DeleteSection, AddSection

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

set_seg_parms

SOURCE

"sfilebl.c"

PURPOSE

Renames Segments and Filter namesets in a Segment file to form the EGS View hierarchy.

SYNOPSIS

```
#include "sfile.h"

int set_seg_parms(fildes, mnum, Segments) ;
    int fildes ;          /* Starbase file descriptor */
    int mnum ;            /* Main Segment Number */
    int Segments[ ] ;     /* List of new Segment Numbers (returned) */
```

DESCRIPTION

This routine allows a 'Display_List' segment number and all subsegment numbers to be changed in the Starbase Display List memory. Also, all of the corresponding filter nameset values are altered to agree with these segment numbers.

In the Real-Time Program this function is called to implement the EGS hierarchy. Each time a new Display List 'View' is loaded into the A or B windows, this function is called to modify the Segment numbers and invisibility filter parameters.

It allows the same 'View' file to be loaded with different switch numbers. The Main Segment Number always corresponds to currently selected View Switch Number.

Returns zero (0) on success, or negative on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

write_display_list

SOURCE

"sfilebl.c"

PURPOSE

Writes 'Display_List' memory to a Starbase Segment ("sfile") format disk file.

SYNOPSIS

```
#include "sfile.h"

int write_display_list(fildes, Pathname) ;
    int    fildes ;          /* Starbase file descriptor */
    char *PathName ;        /* Path/Name of output Segment file */
```

DESCRIPTION

This routine opens and writes a Segment (integer) file from Starbase/Display List memory. It writes all Segments of the currently initialized Starbase device (fildes).

Note: The Starbase file descriptor must previously be initialized.

Returns zero (0) on success or negative on error.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

read_display_list

4.2 ASCII File (asclib.h)

The ASCII file library contains functions which aid in the the handling of the EGS ASCII (text) graphics files. This includes both the ASCII (*.asc) and the older EGS command (*.com) file formats described in Appendices E, G, and H. It is a complete set of **editing** functions which allows ASCII text files to be converted into Starbase/Display List binary files. In addition, routines are provided to read and write **disk I/O** ASCII (*.asc) files. Finally, the line-by-line TTR Command (*.com) to EGS Starbase ASCII (*.asc) **interpreter** (converter) is included with this library. The constants and prototypes for this library are in "asclib.h" (Appendix J).

One important command in the Starbase/Display List ASCII (*.asc) file structure is the Starbase 'application_data' element type. This command allows variable information to be stored in the Starbase Display List. It is used to store header (Type I) and file location (Type II) data. However, an exact number of bytes must be stored and some editors ('vi') do not maintain the correct byte count. These routines handle the 'application_data' byte count properly and should be used instead of relying on a text editor.

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Asc_copy

SOURCE

"asclib2.c"

PURPOSE

To copy a Starbase/Display List 'ascii' buffer in the computer's internal memory while maintaining the correct byte count.

SYNOPSIS

```
#include "acslib.h"

int Asc_copy(In_str, Out_str, In_lines, More) ;
    char In_str[ ][LINE_LENGTH] ; /* Input(*.asc) string buffer */
    char Out_str[ ][LINE_LENGTH] ; /* Output(*.asc) string buffer*/
    int In_lines ; /* Number of lines in Input Buffer */
    int *More ; /* Number of unused lines in Input
    Buffer */
```

DESCRIPTION

This routine copies a 'Display_List' ASCII buffer from an Input String array to an Output String array. Both the Input and Output Buffer arrays must exist before the call and can be of variable length (In_lines). The Output Buffer 'Out_str' must be long enough to hold the complete 'In_str' Buffer.

Returns number of complete 'Display List' lines actually copied to the 'Out_str' Buffer. Any unused lines in the Input buffer are indicated by the 'More' variable (returned).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

read_asc_in, write_asc_out

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

com_to_asc

SOURCE

"asclib1.c"

PURPOSE

Translates older TTR Tektronix 4081 commands into Starbase/Display List ASCII commands.

SYNOPSIS

```
#include "acslib.h"

int com_to_asc(fildes, Com_str, Asc_str, Com_lines, More) ;
    int fildes ;                                /* Starbase graphics file
                                                descriptor */
    char Com_str[ ] [LINE_LENGTH] ; /* Input(*.com) string buffer */
    char Asc_str[ ] [LINE_LENGTH] ; /* Output(*.asc) string buffer*/
    int Com_lines ;                            /* Number of lines in Input
                                                Buffer */
    int *More ;                                /* Number of unused lines in
                                                Input Buffer */
```

DESCRIPTION

This routine copies/translates older TTR Tektronix 4081 commands (from a *.com file) into the ASCII Starbase/Display List commands. The Input Buffer 'Com_str' and Output Buffer 'Asc_str' arrays must exist before the call and can be of variable length (Com_lines). The Output Buffer 'Asc_str' must be long enough to hold the translated contents of 'Com_str' Buffer.

Returns number of complete 'Display List' lines actually translated/copied to the 'Out_str' Buffer. Any unused lines in the Input buffer are indicated by the 'More' variable (returned).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

get_Input_Buf, write_asc_out

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form_DataI

SOURCE

"asclib2.c"

PURPOSE

Formats and fills the Application Data I form with parameter values.

SYNOPSIS

```
#include "acslib.h"

int form_DataI(remark,author,rev,swch,viewname,pathname,
               filename,App_str,app_lines) ;
    char remark[ ] ;                /* Remarks & Comments */
    char author[ ] ;                /* Author's Name */
    int rev ;                       /* Current Revision Number */
    int swch ;                      /* Control Panel Switch # */
    char viewname[ ] ;              /* Control Panel View Name */
    char pathname[ ] ;              /* Path for Filename */
    char filename[ ] ;              /* Filename for Segment or
                                   View */
    char App_str[ ][LINE_LENGTH] ; /* return String Array for
                                   Application element */
    int app_lines ;                 /* Maximum # of lines in
                                   above String Array */
```

DESCRIPTION

This function takes parameter values provided in the parameter list and formats them into the 'application_data' Starbase/ Display List element. It returns the generated ASCII element lines in the 'App_str' array which must exist before the call and can be of variable length (app_lines). The Output Buffer 'App_str' must be long enough to hold the formatted contents (4 lines minimum).

Returns number of lines used in the Application Data array.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form_Hdr, form_DataII

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form_DataII

SOURCE

"asclib2.c"

PURPOSE

Formats and fills the Application Data II form with parameter values.

SYNOPSIS

```
#include "acslib.h"

int form_DataII(filename, App_str, app_lines) ;
    char filename[ ] ;                /* Filename for Segment */
    char App_str[ ][LINE_LENGTH] ;    /* return String Array for
                                       Application element */
    int app_lines ;                   /* Maximum # of lines in
                                       above String Array */
```

DESCRIPTION

This function takes parameter values provided in the parameter list and formats them into the 'application_data' Starbase/Display List element. It returns the generated ASCII element lines in the 'App_str' array which must exist before the call and can be of variable length (app_lines). The Output Buffer 'App_str' must be long enough to hold the formatted contents (2 lines minimum).

Returns number of lines used in the Application Data array.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form_Hdr, form_DataI

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form_Hdr

SOURCE

"asclib2.c"

PURPOSE

Formats and fills the Application Data I form, initiates the filter nameset, and adds the main label.

SYNOPSIS

```
#include "acslib.h"

int form_Hdr(remark,author,rev_num,swch_num,viewname,pathname,
             filename,Rstring,rsize) ;
    char remark[ ] ;           /* Remarks & Comments */
    char author[ ] ;           /* Author's Name */
    int rev_num ;              /* Current Revision Number */
    int swch_num ;             /* Control Panel Switch # */
    char viewname[ ] ;         /* Control Panel View Name */
    char pathname[ ] ;         /* Path for Filename */
    char filename[ ] ;         /* Filename for Segment or
                                View */
    char Rstring[ ][LINE_LENGTH] ; /* return String Array */
    int rsize ;                /* Maximum # of lines in
                                'Rstring' */
```

DESCRIPTION

This function takes parameter values provided in the parameter list and formats them into the 'application_data I' Starbase/Display List element. Also it generates the appropriate filter nameset values and the Main label. It returns the generated lines in the 'Rstring' array which must exist before the call and can be of variable length (rsize). The Return Buffer 'Rstring' must be long enough to hold the formatted contents (13 lines minimum).

The actual number of lines used is returned by the function call.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form_DataI, form_DataII

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

get_Input_Buf

SOURCE

"asclib1.c"

PURPOSE

Reads a buffer of text from Input file and checks for proper formatting (*.com or *.asc).

SYNOPSIS

```
#include "acslib.h"

int get_Input_Buf(fp, iasc, Strng, lines, eof) ;
    FILE *fp ;                /* Input file pointer */
    int iasc ;                 /* Input 'ascii' flag */
    char Strng[ ][LINE_LENGTH] ; /* Return Buffer array */
    int lines ;                /* Maximum Number of lines in
                               above Buffer */
    int *eof ;                 /* End-of-File flag */
```

DESCRIPTION

This function reads a buffer of text from the Input file and checks for the proper formatting.

i.e. **iasc = 0** -> \$ & % for Command (*.com) files.
iasc = 1 -> ; for Starbase ASCII (*.asc) files.

Note only complete command lines are returned.

File (***fp**) must already be open for reading ("r").

The return buffer '**Strng**' must exist and can be of variable length (**lines**).

The End-of-File (***eof**) flag is set (TRUE) if the End of the File is reached.

The actual number of lines filled is returned by the function call.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

open_files, read_asc_in

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

open_files

SOURCE

"asclib1.c"

PURPOSE

Function that gets the Input filename, ASCII output filename, Segment output filename and checks for validity.

SYNOPSIS

```
#include "acslib.h"

int open_files(argc,argv,In,Asc,Sfl,Sfl_name,iasc,oasc,osfl,q) ;
    int  argc ;                /* Number of Command Line
                                Arguments */
    char *argv[ ] ;            /* Pointer to Command Line
                                Arguments */
    FILE **In, **Asc, **Sfl ;  /* File Pointers(returned) */
    char Sfl_name ;            /* Segment filename
                                (*.sfl) returned */
    int  *iasc, *oasc, *osfl ; /* File Flags */
    char *q ;                  /* Quit Flag */
```

DESCRIPTION

This function gets an Input filename, an ASCII output filename, and a Segment output filename from either a Command string (**argc, argv**) or prompts for them from the keyboard.

It then checks for file validity by checking if it exists (**In**) or if can be overwritten (**Asc** or **Sfl**). If everything checks out it opens the files and returns the File Pointers (**In** and **Asc**).

The Segment file is checked for validity but it is not opened on return. However, the filename '**Sfl_name**' is returned for use by the Segment library routines ("sfile.h").

Returned File flag 'iasc' indicates the type of Input file extension as follows:

```
iasc = 0 -> Command (*.com) file.
iasc = 1 -> Starbase ASCII (*.asc) file.
iasc = 2 -> Starbase Segment (*.sfl) file.
```

Returned File flags 'oasc' and 'osfl' indicate whether the particular file output type is desired (and opened - 'Asc').

TRUE (1) -> output file desired.

FALSE(0) -> no output file desired.

The character (q) is the returned quit flag that contains the character 'q' if it is desired to halt program execution.

Returns zero (0) on successful completion or a negative value if an error occurs.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

get_Input_Buf, read_asc_in, write_asc_out, write_display_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

read_asc_in

SOURCE

"asclib2.c"

PURPOSE

Reads a buffer of ASCII text from Input file and checks for proper formatting (*.asc). If a Starbase/Display List 'application_data' element is encountered a binary (byte) transfer is initiated.

SYNOPSIS

```
#include "acslib.h"

int read_asc_in(fp, Strng, lines, eof) ;
FILE *fp ;                      /* Input file pointer */
char Strng[ ][LINE_LENGTH] ; /* Return Buffer array */
int lines ;                     /* Maximum Number of lines in
                                above Buffer */
int *eof ;                     /* End-of-File flag */
```

DESCRIPTION

This function reads a buffer of text from the Input file and checks for the proper formatting.

Note only complete Starbase command lines are returned. Each line must be terminated with a semicolon (;).

File (*fp) must already be open for reading ("r").

The return buffer 'Strng' must exist and can be of variable length (lines).

The End-of-File (*eof) flag is set (TRUE) if the End of the File is reached.

The actual number of lines filled is returned by the function call.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

open_files, get_Input_Buf, write_asc_out

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

write_asc_out

SOURCE

"asclib2.c"

PURPOSE

Writes a buffer of ASCII text to the Output file (*.asc). If a Starbase/Display List 'application_data' element is encountered a binary (byte) transfer is initiated.

SYNOPSIS

```
#include "acslib.h"

int write_asc_out(fp, Strng, lines) ;
    FILE *fp ;                      /* Output file pointer */
    char Strng[ ][LINE_LENGTH] ; /* Output Buffer array to write */
    int lines ;                    /* Maximum Number of lines in
                                   above Buffer */
```

DESCRIPTION

This function writes a buffer of ASCII text to the Output file.

File (*fp) must already be open for writing ("w").

The buffer 'Strng' must exist and can be of variable length (lines).

The actual number of lines written is returned by the function call.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

open_files, get_Input_Buf, read_asc_in

4.3 EGS (egslib.h)

The EGS library contains functions which provide the connection between the EGS real-time process and the HP Starbase library. They initialize all the EGS windows and also are used to format/write the displayed data.

In addition, the HP Fast Alpha/Font Manager library calls are implemented through these library functions.

To facilitate program development and editing, the library was further divided into two parts:

1. Starbase Utilities.
2. Form Utilities.

4.3.1 Starbase Utility

The Starbase Utility library consolidates the Starbase library calls into functions that implement specific EGS tasks.

All of the Starbase Utility functions are contained in the EGS Graphics Real-Time library "egslib.a". The constants and prototypes for this library are in file "egslib.h" (Appendix J).

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

add_hack_lst

SOURCE

"egslib3.c"

PURPOSE

Function which adds a 'hack' element at the end of the Hack linked list.

SYNOPSIS

```
#include "egslib.h"

void add_hack_lst(pos, ls ) ;
    POSITION pos ; /* Current Hack Position to add */
    hk  ls ;      /* Pointer to start of hack list Header */
```

DESCRIPTION

This routine adds a 'hack' element at the end of the Hack Linked List. Dynamically allocates memory as needed.

Utilizes the Hack Structure and the Position Structure.

```
typedef struct Pstn {          /* position structure in TTR */
    float      x, y, z, h ;    /* coordinates. */
} POSITION, *POSITN ;

typedef struct hck {           /* hack structure */
    POSITION    pos ;           /* hack location */
    struct hck *next ;         /* pointer to next hack in list */
} Hack, *hk ;
```

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Write_Hack, Update_Hacks, Clear_Hacks, free_hack_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

add_label_lst

SOURCE

"egslib3.c"

PURPOSE

Function which adds a 'label' element at the end of the Label linked list.

SYNOPSIS

```
#include "egslib.h"

void add_label_lst(pos, time, altitude, ls ) ;
    POSITION pos      /* Current Label Position */
    char time[ ] ;    /* Label Time String */
    char altitude[ ] ; /* Label Altitude String */
    lb ls ;           /* Pointer to start of Label List Header */
```

DESCRIPTION

This routine adds a 'label' element at the end of the Label Linked List. Dynamically allocates memory as needed.

Utilizes the Label Structure and the Position Structure.

```
typedef struct Pstn {           /* position structure in TTR */
    float  x, y, z, h ;         /* coordinates. */
} POSITION, *POSITN ;

typedef struct lbl {           /* Label structure */
    POSITION pos ;               /* Label location */
    char    time[16] ;          /* Label time string */
    char    alt[16] ;           /* Label altitude string */
    struct lbl *next ;          /* pointer to next label in list */
} lbl, *lb ;
```

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Write_Label, Update_Labels, Clear_Labels, free_label_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

add_line_lst

SOURCE

"egslib2.c"

PURPOSE

Function which adds a 'line' element at the end of the Line linked list.

SYNOPSIS

```
#include "egslib.h"

void add_line_lst(pos, md, ls ) ;
    POSITION pos ; /* Current Vertex Position */
    int md ;      /* Move/Draw flag */
    LN ls ;       /* Pointer to start of Line List Header */
```

DESCRIPTION

This routine adds a 'line' element at the end of the Line Linked List. Dynamically allocates memory as needed.

Utilizes the Draw Line Structure and the Position Structure.

```
typedef struct Pstn {          /* position structure in TTR */
    float x, y, z, h ;         /* coordinates. */
} POSITION, *POSITN ;

typedef struct ln {            /* Draw Line structure */
    POSITION pos ;               /* Line Vertex location */
    int md ;                   /* Move/Draw flag */
    struct ln *next ;          /* pointer to next line in list */
} LINE, *LN ;
```

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Draw_Line, free_line_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Checkstring

SOURCE

"egslib3.c"

PURPOSE

Function which compares two strings, ignoring case.

SYNOPSIS

```
#include "egslib.h"

int Checkstring(str1, str2) ;
    char str1[ ] ; /* strings to be compared */
    char str2[ ] ;
```

DESCRIPTION

This routine compares two strings by first converting string#1 to upper case and using the C function 'strcmp'.

Returns **TRUE**(1) if the strings match and **FALSE**(0) if they don't match.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

choice_string

SOURCE

"egslib1.c"

PURPOSE

Function which returns a string from the Starbase keyboard event.

SYNOPSIS

```
#include "egslib.h"

int choice_string(fildes, keyboard, max_len, Strng, dcx, dcy,
    width, height) ;
    int fildes ;           /* Starbase Window file descriptor */
    int keyboard ;         /* Starbase Keyboard file descriptor */
    int max_len ;          /* Maximum string length(characters) */
    char *Strng ;          /* Parsed String returned */
    int dcx, dcy ;         /* current Window cursor location */
    int width, height ;    /* current Window character size */
```

DESCRIPTION

This routine displays a blinking cursor at the specified (pixel) position (**dcx, dcy**), waits for a Starbase keyboard event, parses the pressed key, and saves the character in a buffer string (**Strng**). When an End-of-Line is sensed (CR) the parsed string is returned along with the character count.

The current font size (**width, height**) must be provided (pixels) to determine how much to move the cursor after each key stroke. This information may be obtained with the Font Manager call 'fm_rasterinfo'.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

get_key_command, ProcessKey

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Clear_AER

SOURCE

"egslib2.c"

PURPOSE

Function which clears the Azimuth, Elevation, and Range (AER) window.

SYNOPSIS

```
#include "egslib.h"

void Clear_AER(Awnd, aer, loc, SaveArea, loc_flag, shm_ptr) ;
    int Awnd ;                /* Starbase Tracking Window file
                              descriptor */
    int aer ;                 /* Starbase AER popup Window file
                              descriptor */
    int loc ;                 /* Starbase locator(mouse) file
                              descriptor */
    unsigned char *SaveArea ; /* pointer to background raster
                              save mask */
    int *loc_flag ;           /* active locator process flag */
    master_ptr shm_ptr ;      /* pointer to 'Command' shared memory*/
```

DESCRIPTION

This routine clears the AER pop-up window area and restores the previous background contents. If required, it also turns off the locator (mouse) tracking in the Tracking window.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Display_AER, restore_popup

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Clear_Hacks

SOURCE

"egslib3.c"

PURPOSE

Function which erases all hacks from the screen (A and B windows).

SYNOPSIS

```
#include "egslib.h"
void Clear_Hacks (Awindow, Bwindow, head, shm_ptr) ;
    int Awindow ;          /* Starbase A Window file descriptor */
    int Bwindow ;          /* Starbase B Window file descriptor */
    hk head[ ] ;           /* Pointers to start of 'Hack' Lists */
    master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
```

DESCRIPTION

This routine erases all hacks from the A and B display windows for each active Radar plot. Calls function 'free_hack_list' to free all dynamically allocated Hack Linked List memory and clear the linked list (except the header cell).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Write_Hack, UpDate_Hacks, add_hack_list, free_hack_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Clear_Labels

SOURCE

"egslib3.c"

PURPOSE

Function which erases all labels from the screen (A and B windows).

SYNOPSIS

```
#include "egslib.h"

void Clear_Labels(Awindw, Bwindw, head, shm_ptr) ;
    int  Awindw ;           /* Starbase A Window file descriptor*/
    int  Bwindw ;           /* Starbase B Window file descriptor */
    hk   head ;             /* Pointer to start of 'Label' List */
    master_ptr shm_ptr ;    /* pointer to 'Command' shared memory*/
```

DESCRIPTION

This routine erases all labels from the A and B display windows. Calls function 'free_label_list' to free all dynamically allocated Label Linked List memory and clear the linked list (except the header cell).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Write_Label, UpDate_Labels, add_label_list, free_label_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Clear_REF

SOURCE

"egslib2.c"

PURPOSE

Function which clears the reference point (REF) window.

SYNOPSIS

```
#include "egslib.h"

void Clear_REF(ref, shm_ptr, SaveArea) ;
    int ref ;                /* Starbase REF popup Window file
                             descriptor */
    master_ptr shm_ptr ;     /* pointer to 'Command' shared memory*/
    unsigned char *SaveArea ; /* pointer to background raster
                             save mask */
```

DESCRIPTION

This routine clears the REF pop-up window area and restores the previous background contents.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Display_REF, restore_popup

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

close_graph_devices

SOURCE

"egslib1.c"

PURPOSE

Closes all the Starbase graphics devices opened by function 'init_graph_devices'.

SYNOPSIS

```
#include "egslib.h"

void close_graph_devices (Awindw, Bwindw, Cmd, alphanum, aer, ref,
    keyboard, locator) ;
    int  Awindw, Bwindw ;    /* A & B window file descriptors */
    int  Cmd, alphanum ;    /* Command & Alpha Numeric windows */
    int  aer, ref ;         /* Utility windows AER & REference */
    int  keyboard, locator ; /* Input devices - keyboard & mouse */
```

DESCRIPTION

This routine clears all the graphics display windows, closes all the Starbase graphics devices, and turns on the Alpha ITE screen.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

init_graph_devices

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Concat2D

SOURCE

"egslib1.c"

PURPOSE

Performs a matrix multiply of 3 x 2 matrices.

SYNOPSIS

```
#include "egslib.h"

int Concat2D(MA, MB, MR) ;
    float MA[3][2], MB[3][2] ; /* Input Matrices to Multiply */
    float MR[3][2] ;           /* Resultant return Matrix    */
```

DESCRIPTION

This routine performs a matrix multiply of 3 x 2 matrices.

Before performing the multiply operation each input matrix is converted to a 3 x 3 matrix by function 'Form_33'. The third column is the identity {0, 0, 1}. After the multiplication the resultant matrix MR is converted back to a 3 x 2 format by function 'Form_32'.

Always returns the value zero (0).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Form_33, Form_32, Identity2D, Scale2D, Translate2D, Rotate2D

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Display_AER

SOURCE

"egslib2.c"

PURPOSE

Displays the Azimuth, Elevation, and Range (AER) pop-up window.

SYNOPSIS

```
#include "egslib.h"

void Display_AER(Awnd,aer,mode,fid,loc,tgt_shm,shm_ptr,
  loc_flag,SaveArea,x_val,y_val) ;
  int  Awnd ;          /* Starbase Tracking Window file
                        descriptor */
  int  aer ;           /* Starbase AER popup Window file
                        descriptor */
  int  mode ;          /* Entry mode */
  int  fid ;           /* Font Manager ID */
  int  loc ;           /* Starbase locator(mouse) file
                        descriptor */
  TARGET_SHM tgt_shm ; /* pointer to 'Target' shared memory*/
  master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
  int  *loc_flag ;     /* active locator process flag */
  unsigned char **SaveArea ; /* pointer to background raster
                             Save Area */
  float x_val, y_val ; /* current locator position */
```

DESCRIPTION

This routine displays the AER pop-up window area and formats the Azimuth, Elevation, and Range (AER) data. Uses Font Manager calls to write text information to the AER window.

Operates in one of the following modes:

mode = 0 -> Window Initialization mode.

mode = 1 -> Continuation mode.

mode = 2 -> Screen Update mode.

mode = 3 -> Completion (Clear window) mode.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Clear_AER, save_popup

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Display_REF

SOURCE

"egslib2.c"

PURPOSE

Displays the current reference point value in the (REF) pop-up window.

SYNOPSIS

```
#include "egslib.h"

void Display_REF(ref, fid, tgt_shm, shm_ptr, SaveArea) ;
    int  ref ;                /* Starbase REF popup Window file
                               descriptor */
    int  fid ;                /* Font Manager ID */
    TARGET_SHM tgt_shm ;      /* pointer to 'Target' shared memory*/
    master_ptr shm_ptr ;      /* pointer to 'Command' shared memory*/
    unsigned char **SaveArea ; /* pointer to background raster
                               Save Area */
```

DESCRIPTION

This routine displays the REFERENCE pop-up window area and formats the current reference point coordinates. Uses Font Manager calls to write text information to the REF window.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Clear_REF, save_popup

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Draw_Line

SOURCE

"egslib2.c"

PURPOSE

Draws a line in the A window using either keyboard or locator (mouse) input.

SYNOPSIS

```
#include "egslib.h"
void Draw_Line (Awindow, mode, Wnd, fid, inp, head, shm_ptr, prc_flag,
                SaveArea, x_val, y_val, md) ;
    int  Awindow ;           /* Starbase A Window file descriptor */
    int  mode ;              /* Entry mode */
    int  Wnd ;               /* Starbase popup Window file des. */
    int  fid ;               /* Font Manager ID */
    int  inp ;               /* Input device file descriptor */
    LN  head ;              /* pointer to start of Line List */
    master_ptr shm_ptr ;     /* pointer to 'Command' shared memory */
    int  *prc_flag ;         /* process flag (returned) */
    unsigned char **SaveArea ; /* pointer to Save Area */
    void *x_val, *y_val ;    /* pointer to keyboard value or (x,y)
                               locator values. */
    int  md ;               /* move/draw flag */
```

DESCRIPTION

This routine draws a line segment in the A window between points (vertices). The vertex coordinates may be entered from either the keyboard or locator (mouse) devices.

With each vertex point a move/draw flag (**md**) indicates the graphics operation to be performed to get to that coordinate.

The Draw Line function uses a Line Linked List Structure to store each vertex point value and the number of points is limited only by computer memory.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

add_line_list, free_line_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Form_32

SOURCE

"egslib1.c"

PURPOSE

Forms a (3 x 2) matrix from a (3 x 3) matrix.

SYNOPSIS

```
#include "egslib.h"

void Form_32( M3, M2) ;
    float M3[3][3] ;      /* Input Matrix */
    float M2[3][2] ;      /* Return Matrix */
```

DESCRIPTION

This routine forms a (3 x 2) Matrix from a (3 x 3) Matrix by deleting the last column(3).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Form_33

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Form_33

SOURCE

"egslib1.c"

PURPOSE

Forms a (3 x 3) matrix from a (3 x 2) matrix.

SYNOPSIS

```
#include "egslib.h"

void Form_33 ( M2, M3) ;
    float M2[3][2] ;           /* Input Matrix */
    float M3[3][3] ;           /* Return Matrix */
```

DESCRIPTION

This routine forms a (3 x 3) Matrix from a (3 x 2) Matrix by adding the identity column(3) { 0, 0, 1}.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Form_32

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

free_hack_list

SOURCE

"egslib3.c"

PURPOSE

Frees all memory allocated by 'Hack' linked list.

SYNOPSIS

```
#include "egslib.h"

void free_hack_list(cur) ;
    hk cur ;      /* pointer to start of 'Hack' Linked List */
```

DESCRIPTION

This routine frees all memory allocated by the function 'add_hack_list' except the header element.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

add_hack_list, Write_Hack, UpDate_Hacks, Clear_Hacks

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

free_label_list

SOURCE

"egslib3.c"

PURPOSE

Frees all memory allocated by 'Label' linked list.

SYNOPSIS

```
#include "egslib.h"

void free_label_list(cur) ;
    lb cur ;          /* pointer to start of 'Label' Linked List */
```

DESCRIPTION

This routine frees all memory allocated by the function 'add_label_list' except the header element.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

add_label_list, Write_Label, UpDate_Labels, Clear_Labels

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

free_line_list

SOURCE

"egslib2.c"

PURPOSE

Frees all memory allocated by 'Line' linked list.

SYNOPSIS

```
#include "egslib.h"

void free_line_list(cur) ;
    LN cur ;          /* pointer to start of 'Line' Linked List */
```

DESCRIPTION

This routine frees all memory allocated by the function 'add_line_list' except the header element.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

add_line_list, Draw_Line

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

get_key_command

SOURCE

"egslib1.c"

PURPOSE

Pops up a Prompt Window and gets a Starbase command string from the 'keyboard' input device.

SYNOPSIS

```
#include "egslib.h"

void get_key_command(fildes,keyboard,fid,Prompt,Strng,lines,
                    len,q,X0,Y0,X1,Y1) ;
    int  fildes ;                /* Popup window Starbase file
                                descriptor */
    int  keyboard ;              /* Starbase keyboard file descriptor */
    int  fid ;                   /* Current active Font ID */
    char Prompt[] [LINE_LENGTH] ; /* Window Prompt string */
    char Strng[] [LINE_LENGTH] ; /* Command String (returned) */
    int  lines ;                 /* # of lines in Prompt & Returned
                                Strings */
    int  len ;                   /* Maximum characters in Return Line */
    char *q ;                    /* Quit Flag (returned) */
    float X0, Y0, X1, Y1 ;       /* window coordinates - corners
                                (virtual coordinates) */
```

DESCRIPTION

This routine pops up a window in the '**fildes**' planes, displays a '**Prompt**' string, reads the keyboard, and returns a command string '**Strng**' (or Quit flag '**q**').

More than one Prompt/Command string (**lines**) may be processed in a single call to this function.

Note the function does not return until all Command Strings are processed and may lock the calling program until it is finished.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

save_popup, restore_popup, choice_string, ProcessKey

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Get_RefIn

SOURCE

"egslib2.c"

PURPOSE

Pops up a Prompt Window and gets a new Reference point from the keyboard input device.

SYNOPSIS

```
#include "egslib.h"

void Get_RefIn(Cmd, fid, mode, shm_ptr, ref_num, key_flag, SaveArea,
               key_val) ;
    int  Cmd ;                /* Popup window Starbase file descriptor */
    int  fid ;                /* Font Manager ID */
    int  mode ;               /* Entry mode */
    master_ptr shm_ptr ;      /* pointer to 'Command' shared memory*/
    int  *ref_num ;           /* new reference number (returned) */
    int  *key_flag ;          /* process flag (returned) */
    unsigned char **SaveArea ; /* pointer to popup Save Area */
    int  key_val ;            /* current key value */
```

DESCRIPTION

This routine pops up a window in the 'Cmd' planes, displays a Prompt string, reads the key value, and returns the reference coordinates to 'Command' shared memory.

Note the function returns after each key value is processed and does not lock the calling program.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

save_popup, restore_popup, ProcessKey, parse_lat_lng

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Identity2D

SOURCE

"egslib1.c"

PURPOSE

Forms a two-dimensional Identity Matrix (3 x 2).

SYNOPSIS

```
#include "egslib.h"

int Identity2D(M) ;
    float M[3][2] ;          /* Return Matrix */
```

DESCRIPTION

This routine forms a two-dimensional Identity Matrix (3 x 2).

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Scale2D, Translate2D, Rotate2D, Concat2D

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

init_Font_Manager

SOURCE

"egslib1.c"

PURPOSE

Sets the window Font Manager parameters and defaults.

SYNOPSIS

```
#include "egslib.h"

void init_Font_Manager(fildes, font_path, fid, bcolor, fcolor) ;
    int  fildes ;           /* window file descriptor */
    char *font_path ;       /* Font Manager font (path) to load */
    int  *fid ;             /* font identification (returned) */
    int  bcolor ;           /* background color index */
    int  fcolor ;           /* foreground color index */
```

DESCRIPTION

This routine initializes a window's Font Manager font and colors.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

init_graph_devices

SOURCE

"egslib1.c"

PURPOSE

Initializes all the EGS Starbase devices.

SYNOPSIS

```
#include "egslib.h"

void init_graph_devices(wnd_plane, Awindow, Bwindow, Cmd, alphanum,
                       aer, ref, keyboard, locator) ;
    int  wnd_plane[ ] ;      /* planes array (image/overlay) */
    int  *Awindow, *Bwindow ; /* A & B window file descriptors
                               (returned) */
    int  *Cmd, *alphanum ;   /* Command & Alpha Numeric window
                               file descriptors (returned) */
    int  *aer, *ref ;        /* Utility windows - AER & REF
                               file descriptors (returned) */
    int  *keyboard ;         /* Input device - keyboard
                               file descriptor (returned) */
    int  *locator ;          /* Input device - mouse
                               file descriptor (returned) */
```

DESCRIPTION

This routine initializes all the Starbase graphics devices and determines in which display planes 'wnd_planes' they are loaded.

Uses the following environmental variables when initializing the graphics devices (default).

"SB_OUTDEV"	='/dev/crt'	"SB_OUTDRIVER"	='hp98550'
"SB_OVDEV"	='/dev/ocrt'	"SB_OVDRIVER"	='hp98550'
"SB_KB"	='/dev/tty2'	"SB_KBDRIVER"	='keyboard'
"SB_INDEV"	='/dev/locator'	"SB_INDRIVER"	='hp-hil'

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

init_window

SOURCE

"egslib1.c"

PURPOSE

Initializes a Starbase graphics window's parameters (defaults).

SYNOPSIS

```
#include "egslib.h"

void init_window(fildes, vdcx0, vdcy0, vdcx1, vdcy1, wcx0, wcy0,
                 wcx1, wcy1, map_mode) ;
    int    fildes ;           /* window file descriptor */
    float  vdcx0, vdcy0 ;     /* virtual coordinates - lower left */
    float  vdcx1, vdcy1 ;     /* virtual coordinates - upper right */
    float  wcx0, wcy0 ;       /* world coordinates - lower left */
    float  wcx1, wcy1 ;       /* world coordinates - upper right */
    int    map_mode ;         /* sent to 'mapping_mode' */
```

DESCRIPTION

This routine initializes a Starbase graphics window's default parameters as follows.

aspect ratio	- x(0.0 -> 1.25) & y(0.0 -> 1.0)	
viewport	- virtual coordinate parameters (vdcx,vdcy)	
	world coordinate parameters (wcx, wcy)	
mapping mode	- 'map_mode'	
clipping	- CLIP_TO_VIEWPORT	
buffer mode	- On	
interior style	- INT_HALLOW	
highlighting	- Off	
background color	- Black	
line color	- White	perimeter color - White
fill color	- Black	highlight color - White
marker color	- Gray50	text color - White
marker type	- dot	marker size - .02 * vdcy1
text type	- STROKE_TEXT	character size - .02 * vdcy1
character set	- "usascii"	
text alignment	- HALF - LEFT	

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

parse_lat_lng

SOURCE

"egslib3.c"

PURPOSE

Parses a Source String which contains a Latitude and/or Longitude sub-string(s).

SYNOPSIS

```
#include "egslib.h"

int parse_lat_lng(Strng, des_str, sp, slen, max) ;
    char Strng[ ] ;    /* Source String */
    char des_str[ ] ;  /* Destination String (returned) */
    int  *sp ;         /* string location; start sub-string */
    int  slen ;        /* Source string length */
    int  max ;         /* Maximum Number of characters in Destination
                        string */
```

DESCRIPTION

This routine parses a Source string for a Latitude (Longitude) sub-string starting with character 'sp'. Formats and returns parsed string in 'des_str'.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Get_RefIn

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Plot_Radar_Sym

SOURCE

"egslib2.c"

PURPOSE

Plots a Radar Tracking Symbol in both the A and B windows.

SYNOPSIS

```
#include "egslib.h"

void Plot_Radar_Sym(plot_sym, Awindw, Bwindw, radar_num, shm_ptr,
                    tgt_ptr, Trail, next, bflag) ;
    int plot_sym ;           /* Plot symbol (marker) to use */
    int Awindw, Bwindw ;     /* A & B window file descriptors */
    int radar_num ;          /* Radar number to plot */
    master_ptr shm_ptr ;     /* pointer to 'Command' shared memory */
    TARGET_SHM *tgt_ptr ;    /* pointer to 'Target' shared memory */
    POSITION Trail[ ] ;       /* Radar Trail position array */
    int next ;               /* Radar Trail array index flag */
    int bflag ;              /* Marker Blink flag */
```

DESCRIPTION

This routine erases the old Radar tracking symbol and draws a new Radar symbol '**plot_sym**' at the current radar location. It also draws a Trail symbol at the old tracking position and erases the oldest Trail point. Finally it updates the Trail position array ('**Trail**') for the current Radar. It plots the Radar symbol and Trail symbol in both the A and B windows.

It does not update the trail pointer, but leaves it alone in case other Radar Symbols needed plotting in subsequent function calls.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Plot_Symbol

SOURCE

"egslib3.c"

PURPOSE

Plots a Symbol in both the A and B windows.

SYNOPSIS

```
#include "egslib.h"

void Plot_Symbol(Awindw,Bwindw,num,mode,tgt_shm,shm_ptr) ;
    int Awindw, Bwindw ; /* A & B window file descriptors */
    int num ;             /* Plot symbol (marker) to use */
    int *mode ;           /* Function Mode - Draw(0)/Erase(1) */
    TARGET_SHM *tgt_shm ; /* pointer to 'Target' shared memory */
    master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
```

DESCRIPTION

This routine plots one of two markers in the A and B windows at the position specified in the 'Command' shared memory. It uses the default symbols (colors) for 'IN1' & 'IN2' defined in file "egslib.h".

If 'mode' equals zero(0) the Draw Symbol mode is active otherwise the Erase Symbol mode ('mode' not zero(0)) is active.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

ProcessKey

SOURCE

"egslib1.c"

PURPOSE

Processes a keyboard input value.

SYNOPSIS

```
#include "egslib.h"

void ProcessKey(fildes, key, Strng, count, x, y, cw, ch, eol) ;
    int  fildes ;      /* Display Window file descriptor */
    int  key ;         /* keyboard input value */
    char *Strng ;      /* String value (returned) */
    int  *count ;      /* current character count (updated) */
    int  *x, *y ;      /* current screen location (dc's) */
    int  cw, ch ;      /* current character size */
    int  *eol ;        /* End-of-line flag (returned) */
```

DESCRIPTION

This routine takes a keyboard input value 'key', interprets the character, and performs the appropriate action. When a <CR> {'/' or '/n'} is detected the End-of-line 'eol' flag is set. The return 'Strng' contains the partial string as it exists or the complete string if the 'eol' flag is set.

The current screen location 'x, y' is the current display screen position for the next write. It is updated by this program to reflect any changes made.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

get_key_command, choice_string, Get_RefIn

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Remove_Com_Shm

SOURCE

"egslib3.c"

PURPOSE

Removes the 'Command' shared memory block.

SYNOPSIS

```
#include "egslib.h"

void Remove_Com_Shm(shmid, semid) ;
    int  shmid ;    /* shared memory id number */
    int  semid ;    /* semaphore id number */
```

DESCRIPTION

This routine removes the 'Command' shared memory block from memory. Normally this function is just used for debugging purposes.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

restore_popup

SOURCE

"egslib1.c"

PURPOSE

Restores the current background raster memory in the specified Starbase window.

SYNOPSIS

```
#include "egslib.h"

void restore_popup(fildes, memory, x0, y0, x1, y1) ;
    int  fildes ;           /* Window file descriptor */
    unsigned char *memory ; /* pointer to restoration memory*/
    int  x0, y0 ;           /* lower-left corner of window(dc's) */
    int  x1, y1 ;           /* upper-right corner of window(dc's)*/
```

DESCRIPTION

This routine restores the specified Starbase window background raster memory and then frees the dynamically allocated memory block.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

save_popup

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Rotate2D

SOURCE

"egslib1.c"

PURPOSE

Forms a two-dimensional Rotation Matrix (3 x 2).

SYNOPSIS

```
#include "egslib.h"

int Rotate2D(M, Theta) ;
    float M[3][2] ;    /* Return Matrix */
    float Theta ;      /* Amount of rotation (degrees) */
```

DESCRIPTION

This routine forms a two-dimensional Rotation Matrix (3 x 2) from the rotation angle 'Theta'.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Scale2D, Translate2D, Identity2D, Concat2D

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

save_popup

SOURCE

"egslib1.c"

PURPOSE

Saves the specified Starbase window background raster memory in dynamically allocated program memory. Clears the window to the specified background color.

SYNOPSIS

```
#include "egslib.h"

unsigned char *save_popup(fildes, x0, y0, x1, y1, bgcolor) ;
    int  fildes ; /* Window file descriptor */
    int  x0, y0 ; /* lower-left corner of window(dc's) */
    int  x1, y1 ; /* upper-right corner of window(dc's) */
    int  bgcolor ; /* background color index */
```

DESCRIPTION

This routine saves the specified Starbase window background raster memory in a dynamically allocated memory block. Also, it clears the window area to the specified background color.

Returns a pointer to the memory block allocated with the function 'malloc'.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

restore_popup

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Scale2D

SOURCE

"egslib1.c"

PURPOSE

Forms a two-dimensional Scaling Matrix (3 x 2).

SYNOPSIS

```
#include "egslib.h"

int Scale2D(M, Sx, Sy) ;
    float M[3][2] ; /* Return Matrix */
    float Sx ;      /* Scale Factor for X direction */
    float Sy ;      /* Scale Factor for Y direction */
```

DESCRIPTION

This routine forms a two-dimensional Scaling Matrix (3 x 2) from the X and Y scale factors.

Note to prevent distortion the X and Y scale factors must be the same.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Identity2D, Translate2D, Rotate2D, Concat2D

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

SetCom_defaults

SOURCE

"egslib3.c"

PURPOSE

Sets default values in 'Command' shared memory.

SYNOPSIS

```
#include "egslib.h"
void SetCom_defaults(shm_ptr, semid) ;
    master_ptr shm_ptr    /* pointer to 'Command' shared memory*/
    int          semid ;   /* semaphore id number */
```

DESCRIPTION

This routine sets reasonable default values for every location in 'Command' shared memory. Usually called at program start-up to initialize the 'Command' shared memory to usable values.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Remove_Com_Shmem

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Set_CTM

SOURCE

"egslib1.c"

PURPOSE

Sets the the Current Transformation Matrix (CTM) for the specified Starbase Graphics window.

SYNOPSIS

```
#include "egslib.h"

void Set_CTM( Wnd, Tx, Ty, S, Theta) ;
    int    Wnd ;          /* Window file descriptor */
    float Tx, Ty ;        /* X & Y Translation values */
    float S ;             /* X & Y Scale Factor */
    float Theta ;         /* Rotation value (degrees) */
```

DESCRIPTION

This routine sets the Current Transformation Matrix (CTM) for the specified Starbase Graphics window ('Wnd'). This results in Panning ('Tx, Ty'), Zooming ('S'), or Rotating ('Theta') the displayed View.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Identity2D, Scale2D, Translate2D, Rotate2D, Concat2D

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Set_Softkeys

SOURCE

"egslib1.c"

PURPOSE

Sets the User Softkeys (Function keys) to the Label Names and return key characters specified in file "egslib.h".

SYNOPSIS

```
#include "egslib.h"

void Set_Softkeys() ;
```

DESCRIPTION

This routine programs the User Softkeys (Function keys) to the Label Names and key characters defined in the header file "egslib.h".

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Translate2D

SOURCE

"egslib1.c"

PURPOSE

Forms a 2 dimensional Translation Matrix (3 x 2).

SYNOPSIS

```
#include "egslib.h"

int Translate2D(M, Tx, Ty) ;
    float M[3][2] ;    /* Return Matrix */
    float Tx ;          /* Translation in X direction */
    float Ty ;          /* Translation in Y direction */
```

DESCRIPTION

This routine forms a two-dimensional Translation Matrix (3 x 2) from the X and Y distance factors in meters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Identity2D, Scale2D, Rotate2D, Concat2D

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Update_Hacks

SOURCE

"egslib3.c"

PURPOSE

Updates Screen Hacks from data in Hack List on both the A and B windows (after a screen refresh).

SYNOPSIS

```
#include "egslib.h"

void Update_Hacks(hack_sym, Awindw, Bwindw, shm_ptr, head) ;
    int  hack_sym ;          /* Plot symbol (marker) to use */
    int  Awindw, Bwindw ;    /* A & B window file descriptors */
    master_ptr shm_ptr ;     /* pointer to 'Command' shared memory*/
    hk   head ;              /* pointer to start of 'Hack' List */
```

DESCRIPTION

This routine updates the display screen (A and B windows) hacks from POSITION parameters in the Hack list. It handles one radar per call.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Write_Hack, Clear_Hacks, add_hack_list, free_hack_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Update_Labels

SOURCE

"egslib3.c"

PURPOSE

Updates Screen Labels from data in Label List on both the A and B windows (after a screen refresh).

SYNOPSIS

```
#include "egslib.h"

void Update_Labels( Awindw, Bwindw, head, shm_ptr) ;
    int  Awindw, Bwindw ; /* A & B window file descriptors */
    lb   head ;           /* pointer to start of 'Label' List */
    master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
```

DESCRIPTION

This routine updates the display screen (A and B windows) labels from POSITION, Time, and Altitude parameters in the Label list.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Write_Label, Clear_Labels, add_label_list, free_label_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Update_Trail

SOURCE

"egslib2.c"

PURPOSE

Updates Screen Trail Symbols from data in Trail Array on both the A and B windows (after a screen refresh).

SYNOPSIS

```
#include "egslib.h"

void Update_Trail(Awindw,Bwindw,shm_ptr,Trail,next_trail,s_size,
                  s_color,s_type) ;
    int  Awindw, Bwindw ; /* A and B window file descriptors */
    master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
    POSITION Trail[ ] ; /* Trail position array (one radar) */
    int  next_trail ; /* current trail position (index) */
    float s_size ; /* trail symbol size */
    int  s_color ; /* trail symbol color index */
    int  s_type ; /* trail symbol (marker) type */
```

DESCRIPTION

This routine updates the display screen (A and B windows) trail markers from POSITION coordinates in the Radar Trail array '**Trail**'.

Only one Radar trail is updated per function call.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Plot_Radar_Sym

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Write_Hack

SOURCE

"egslib3.c"

PURPOSE

Writes a Hack symbol in the A and B windows for all active radars.

SYNOPSIS

```
#include "egslib.h"

void Write_hack(Awindow, Bwindow, head, tgt_shm, shm_ptr) ;
    int  Awindow, Bwindow ; /* A & B window file descriptors */
    hk   head[ ] ;          /* array of pointers to start of 'Hack'
                             Lists (each radar) */
    TARGET_SHM *tgt_ptr ; /* pointer to 'Target' shared memory */
    master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
```

DESCRIPTION

This routine writes a Hack symbol for each radar in the A and B windows at the current radar coordinates. It also adds the hack location to the 'Hack' List for each Radar.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Update_Hacks, Clear_Hacks, add_hack_list, free_hack_list

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

Write_Label

SOURCE

"egslib3.c"

PURPOSE

Writes a marker, time, and altitude label in the A and B windows for Radar A only.

SYNOPSIS

```
#include "egslib.h"

void Write_Label(Awindow, Bwindow, head, shm_ptr, tgt_shm) ;
    int Awindow, Bwindow ; /* A & B window file descriptors */
    lb head ; /* pointer to start of 'Label' List */
    master_ptr shm_ptr ; /* pointer to 'Command' shared memory*/
    TARGET_SHM *tgt_ptr ; /* pointer to 'Target' shared memory */
```

DESCRIPTION

This routine writes a marker, time, and altitude string in the A and B windows at the current Radar A coordinates. It also adds the label location to the 'Label' List.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

Update_Labels, Clear_Labels, add_label_list, free_label_list

4.3.2 Form Utility

The Form Utility library contains the HP Fast Alpha/Font Manager library calls which implement the AlphaNumeric window display.

All of the Form Utility functions are placed in a separate object file "fmlib1.o". This was done to allow the addition of future FORMS without recompiling the complete EGS Graphics Real-Time library. The constants and prototypes for this library are in file "egslib.h" (Appendix J).

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form3

SOURCE

"frmlib1.c"

PURPOSE

Function which displays the background (non-changing) information for the Air Drop AlphaNumeric Window display.

SYNOPSIS

```
#include "egslib.h"
void form3(gfd, fid, vdcx0, vdcy0, vdcx1, vdcy1) ;
    int gfd ; /* AlphaNumeric window file descriptor */
    int fid ; /* Font Manager Font ID to use */
    float vdcx0, vdcy0 ; /*lower-left corner of window(vdc's) */
    float vdcx1, vdcy1 ; /* upper-right corner of window(vdc's)*/
```

DESCRIPTION

This routine draws the background portion of the AlphaNumeric window. This is the static (non-changing) portion of the window and only needs to be called when the complete screen is refreshed. It uses Font Manager library function calls to display title information at specific locations in the window. It also saves the write locations for following calls by the function 'form3_fill'.

The following information is displayed in this FORM.

Radar A and B Names
Reference Point Name, location(X,Y,& H), and Course Line
Radar A Track Location (X, Y, and H) and Time to Target
Display Units (feet, meters, or Nautical Miles)
Date and Time
Operation Number
Target A Magnetic Heading, Altitude, and Velocity
Target B Magnetic Heading, Altitude, and Velocity
Target A to Target B Bearing and Range

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form3_fill

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form3_fill

SOURCE

"frmlib1.c"

PURPOSE

Function which calculates and displays the time varying information for the Air Drop AlphaNumeric Window display.

SYNOPSIS

```
#include "egslib.h"

void form3_fill(gfd, fid, tgt_shm_ptr, com_shm_ptr) ;
    int gfd ;                /* AlphaNumeric window file descriptor */
    int fid ;                /* Font Manager Font ID to use */
    TARGET_SHM *tgt_shm_ptr ; /* pointer to 'Target' shared memory */
    master_ptr com_shm_ptr ; /* pointer to 'Command' shared memory */
```

DESCRIPTION

This routine performs calculations and displays the time varying data in the AlphaNumeric window. Its update rate is the alpha string update rate (1 per second). It uses Font Manager library calls to display data information at the screen locations passed from the function 'form3'.

The following data displayed by this FORM.

A and B Names

Reference Point Name, location(X,Y,& H), and Course Line

Radar A Track Location (X, Y, and H) and Time to Target

Display Units (feet, meters, or Nautical Miles)

Date and Time

Operation Number

Target A Magnetic Heading, Altitude, and Velocity

Target B Magnetic Heading, Altitude, and Velocity

Target A to Target B Bearing and Range

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form3

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form4

SOURCE

"frmlib1.c"

PURPOSE

Function which displays the background (non-changing) information for the Blank AlphaNumeric Window display.

SYNOPSIS

```
#include "egslib.h"

void form4(gfd, fid, vdcx0, vdcy0, vdcx1, vdcy1) ;
    int  gfd ;          /* AlphaNumeric window file descriptor */
    int  fid ;          /* Font Manager Font ID to use */
    float vdcx0, vdcy0 ; /* lower-left corner of window(vdc's) */
    float vdcx1, vdcy1 ; /* upper-right corner of window(vdc's) */
```

DESCRIPTION

This routine draws the background portion of the Blank AlphaNumeric window.

The following information is displayed in this FORM.

None

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form4_fill

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

form4_fill

SOURCE

"frmlib1.c"

PURPOSE

Function which completes the Blank Window display function call.

SYNOPSIS

```
#include "egslib.h"
void form4_fill(gfd, fid, tgt_shm_ptr, com_shm_ptr) ;
    int gfd ;                /* AlphaNumeric window file descriptor */
    int fid ;                /* Font Manager Font ID to use */
    TARGET_SHM *tgt_shm_ptr ; /* pointer to 'Target' shared memory */
    master_ptr com_shm_ptr ; /* pointer to 'Command' shared memory */
```

DESCRIPTION

This routine completes the Blank window display function calls.

The following data displayed by this FORM.

None

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

form4

4.4 TTR Coordinate Conversion (ttrcoord.h)

The TTR Coordinate Transformation library contains functions which convert between various coordinate systems. In addition, several utility functions are also included which are used in the Graphics Real-Time Program (GRTP). These coordinate system conversions are of the following general types:

1. TTR Cartesian Coordinates (x,y,z)
2. TTR Spherical Coordinates (Azimuth, Elevation, and Range) from a Reference Point (Range zero)
3. Earth-Centered Cartesian Coordinates (E, F, and G)
4. Global Coordinate System (Latitude, Longitude, and Height)

Functions for global conversions are based on one of three ellipsoids (datums):

1. North American Datum of 1927 (NAD27)
2. WGS72 ellipsoid coordinate system
3. WGS84 ellipsoid coordinate system

Appendix F contains a mathematical description of the equations used in developing these library functions (NAD27).

Also, Appendix F lists the Molodensky datum transformation equations that are used to convert WGS84 coordinate values into NAD27 coordinate values (Latitude, Longitude, and Height).

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

conv_magn

SOURCE

"rngcoord.c"

PURPOSE

Adjusts the Azimuth from true north to magnetic north at TTR.

SYNOPSIS

```
#include "ttrcoord.h"

double conv_magn(Az) ;
double Az ;          /* True North Azimuth (degrees) */
```

DESCRIPTION

This routine takes the true north azimuth value "Az" and adds the Magnetic Declination (MDCL) value.

Returns the Magnetic (Compass) Heading.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

conv_truen

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

conv_truen

SOURCE

"rngcoord.c"

PURPOSE

Adjusts the Azimuth from magnetic north to true north at TTR.

SYNOPSIS

```
#include "ttrcoord.h"

double conv_truen(Az) ;
    double Az ;                /* Magnetic North Azimuth (degrees) */
```

DESCRIPTION

This routine takes the magnetic north azimuth value "Az" and subtracts the Magnetic Declination (MDCL) value.

Returns the True North Heading from the compass (magnetic) heading.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

conv_magn

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

deg_sdms

SOURCE

"ttrcoord.c"

PURPOSE

Takes a double precision floating point value (degrees) and converts it to a "Degrees Minutes'Seconds" string.

SYNOPSIS

```
#include "ttrcoord.h"

void deg_sdms(deg, Strng) ;
    double deg ;          /* degrees - decimal */
    char *Strng ;         /* String returned - dd mm'ss.sssss" */
```

DESCRIPTION

This routine takes the double precision value "deg" and formats it into a character string "Strng".

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

sdms_deg

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

llh

SOURCE

"ttrcoord.c"

PURPOSE

Gets geodetic Latitude, Longitude, and Height coordinates (NAD27) from an earth-centered coordinate system. Uses iterative solution.

SYNOPSIS

```
#include "ttrcoord.h"

void llh( E, F, G, xlat, elong, hght) ;
    double E, F, G ; /* Earth Centered coordinates */
    double *xlat ; /* Geodetic Latitude (degrees) */
    double *elong ; /* Longitude (degrees) */
    double *hght ; /* Height above Mean Sea Level (MSL) */
```

DESCRIPTION

This routine takes earth-centered coordinates ('E, F, G') and converts them to Latitude ('xlat'), Longitude ('elong'), and Height ('hght') values.

It uses the NAD27 datum and computes an iterative solution.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

xyz

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

llh72

SOURCE

"ttrcoord.c"

PURPOSE

Gets geodetic Latitude, Longitude, and Height coordinates (WGS72) from an earth-centered coordinate system. Uses iterative solution.

SYNOPSIS

```
#include "ttrcoord.h"

void llh72( E, F, G, xlat, elong, hght) ;
    double E, F, G ; /* Earth Centered coordinates */
    double *xlat ; /* Geodetic Latitude (degrees) */
    double *elong ; /* Longitude (degrees) */
    double *hght ; /* Height above Mean Sea Level (MSL) */
```

DESCRIPTION

This routine takes earth-centered coordinates ('E, F, G') and converts them to Latitude ('xlat'), Longitude ('elong'), and Height ('hght') values.

It uses the WGS72 datum and computes an iterative solution.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

xyz72

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

molod

SOURCE

"molod.c"

PURPOSE

Routine to execute the Molodensky datum transformation per Defense Mapping Agency Technical Report 8350.2.

SYNOPSIS

```
#include "ttrcoord.h"

void molod(lat, lon, h, a, f, esq, da, df, dx, dy, dz, &dlat, &dlon, &dh)
    double lat ;      /* "from" geodetic latitude, radians. */
    double lon ;      /* "from" geodetic longitude, radians. */
    double h ;        /* "from" height (ellipsoid), meters. */
    double a ;        /* Semi-major axis, meters, "from" ellipsoid */
    double f ;        /* Flattening of "from" ellipsoid. */
    double esq ;      /* Eccentricity squared - "from" ellipsoid */
    double da ;      /* Delta of Semi-major axis, "to" minus "from" ellipsoid. */
    double df ;      /* Delta of flattening, "to" minus "from" ellipsoid. */
    double dx, dy, dz /* Translations, meters. */
    double *dlat ;    /* Delta latitude, radians. */
    double *dlon ;    /* Delta longitude, radians. */
    double *dh ;      /* Delta in height, meters. */
```

DESCRIPTION

This routine takes World coordinates (Latitude, Longitude, and Height) "from" one Geodetic Datum (NAD27) and converts them "to" another World coordinate Datum (WGS84) Latitude, Longitude, and Height values.

The output arguments (dlat, dlon, and dh) are changes to be added to "from" coordinates to derive "to" coordinates.

AUTHOR

D. F. Anderson, SNL - 2719

SEE ALSO

wgsnad

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

rae_xyz

SOURCE

"rngcoord.c"

PURPOSE

Converts slant Range, Azimuth, and Elevation to TTR range Cartesian coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void rae_xyz( Rg, Az, El, x, y, z) ;
    double Rg, Az, El ; /* TTR - Range, Azimuth, & Elevation */
    double *x, *y, *z ; /* TTR range Cartesian coordinates */
```

DESCRIPTION

This routine takes slant Range ('Rg'), Azimuth ('Az'), and Elevation ('El') values and converts them into TTR range Cartesian coordinates ('x, y, z').

The Azimuth and Elevation are in units of degrees. With zero(0) degrees referenced to North (Azimuth) and zero(0) degrees horizontal for Elevation.

The 'x, y, z' units are identical to the Range (Rg) units value.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

xyz_rae

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

rotat1

SOURCE

"ttrcoord.c"

PURPOSE

Takes local TTR coordinates and rotates them into a system that is parallel to earth-centered coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void rotat1(vnorth, veast, vout, xlat, elong, x, y, z) ;
    double vnorth ;      /* TTR range - local north  (y) */
    double veast ;       /* TTR range - local east   (x) */
    double vout ;        /* TTR range - local height (z) */
    double xlat ;        /* geodetic latitude (degrees) */
    double elong ;       /* longitude (degrees)      */
    double *x, *y, *z ;  /* rotated coordinates (returned) */
```

DESCRIPTION

This routine takes local TTR coordinates and rotates them to conform to a coordinate system that is parallel to an earth-centered system.

The TTR range zero geodetic latitude "**xlat**" and longitude "**elong**" must be supplied as referenced to the datum (NAD27 or WGS72) that you are interested in converting.

TTR Range zero (NAD27)	- ZLAT27 =	37.84353078 degrees
	ZLNG27 =	-116.72898419 degrees

TTR Range zero (WGS72)	- ZLAT72 =	37.84343292 degrees
	ZLNG72 =	-116.73004621 degrees

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

rotat2

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

rotat2

SOURCE

"ttrcoord.c"

PURPOSE

Takes earth-centered coordinates and rotates them into a system that is parallel to the local TTR system.

SYNOPSIS

```
#include "ttrcoord.h"

void rotat2( x, y, z, vnorth, veast, vout, xlat, elong) ;
    double x ;           /* Earth - local east      (x) */
    double y ;           /* Earth - local north   (y) */
    double z ;           /* Earth - local height  (z) */
    double *vnorth ;     /* TTR range - local north (y) */
    double *veast ;      /* TTR range - local east  (x) */
    double *vout ;       /* TTR range - local height (z) */
    double xlat ;        /* geodetic latitude (degrees) */
    double elong ;       /* longitude (degrees) */
```

DESCRIPTION

This routine takes earth-centered coordinates and rotates them to form a coordinate system that is parallel to the local TTR range system.

The TTR range zero geodetic latitude "**xlat**" and longitude "**elong**" must be supplied as referenced to the datum (NAD27 or WGS72) that you are interested in converting.

TTR Range zero (NAD27) -	ZLAT27 =	37.84353078 degrees
	ZLNG27 =	-116.72898419 degrees

TTR Range zero (WGS72) -	ZLAT72 =	37.84343292 degrees
	ZLNG72 =	-116.73004621 degrees

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

rotat1

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

rotate_cl

SOURCE

"rngcoord.c"

PURPOSE

Translates the origin to the current reference point and rotates the normal (x,y) Cartesian system such that the 'y-axis' now lies along the course line.

SYNOPSIS

```
#include "ttrcoord.h"

void rotate_cl( cl, xref, yref, x, y, xp, yp) ;
    double cl ;          /* current course line (degrees) */
    double xref, yref ;  /* current reference point */
    double x, y ;        /* current coordinate values */
    double *xp, *yp ;    /* rotated/translated values (returned) */
```

DESCRIPTION

This routine translates the current TTR coordinates ('x,y') such that they appear to reside in a new Cartesian system. The new origin is 'xref, yref' and the new 'y-axis' lies along the course line ('cl').

The course line is expressed in degrees with zero(0) degrees due North and the plus (+) direction clockwise.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

time_to_go

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

sdms_deg

SOURCE

"ttrcoord.c"

PURPOSE

Takes a "Degrees Minutes'Seconds" string and converts it to a double precision floating point value (degrees).

SYNOPSIS

```
#include "ttrcoord.h"

double sdms_deg(String) ;
    char *String ;    /* String to be parsed - dd mm'ss.sssss" */
```

DESCRIPTION

This routine takes the character string "**String**", parses it for the correct format, and converts it to a double precision value.

It returns this double precision value in units of degrees.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

deg_sdms

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

tearth_rng

SOURCE

"ttrcoord.c"

PURPOSE

Performs a linear translation of earth-centered coordinates (NAD27) to local range coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void tearth_rng( x, y, z, E, F, G)
    double *x, *y, *z ; /* local range coordinates */
    double E, F, G ;    /* Earth centered coordinates */
```

DESCRIPTION

This routine does a linear translation of earth-centered coordinates (Greenwich) to TTR range coordinates. Note it does not perform any coordinate rotations. Use 'rotat2' for that function.

It assumes a NAD27 datum for translation parameters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

trng_earth

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

tearth_rng72

SOURCE

"ttrcoord.c"

PURPOSE

Performs a linear translation of earth-centered coordinates (WGS72) to local range coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void tearth_rng72( x, y, z, E, F, G)
    double *x, *y, *z ; /* local range coordinates */
    double E, F, G ;     /* Earth centered coordinates */
```

DESCRIPTION

This routine does a linear translation of earth-centered coordinates (Greenwich) to TTR range coordinates. Note it does not perform any coordinate rotations. Use 'rotat2' for that function.

It assumes a WGS72 datum for translation parameters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

trng_earth72

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

time_to_go

SOURCE

"rngcoord.c"

PURPOSE

Calculates the amount of time remaining for a radar-tracked object to arrive at the reference point.

SYNOPSIS

```
#include "ttrcoord.h"

double time_to_go( yp, vel) ;
    double yp ;      /* distance to target */
    double vel ;     /* current velocity of object */
```

DESCRIPTION

This routine calculates the amount of time remaining for an object to arrive at the reference point (target) when it is traveling at a ground velocity of 'vel'.

It is based on the distance from the target (reference point) 'yp'. Coordinate 'yp' must lay on the course line and represents the distance to the target. It is calculated by using function 'rotate_cl'.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

rotate_cl

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

trng_earth

SOURCE

"ttrcoord.c"

PURPOSE

Performs a linear translation of local range coordinates to earth-centered coordinates (NAD27).

SYNOPSIS

```
#include "ttrcoord.h"

void trng_earth( E, F, G, x, y, z)
    double *E, *F, *G ;    /* Earth centered coordinates */
    double x, y, z ;        /* local range coordinates */
```

DESCRIPTION

This routine does a linear translation of TTR range coordinates to earth-centered coordinates (Greenwich). Note it does not perform any coordinate rotations. Use 'rotat1' for that function.

It assumes a NAD27 datum for translation parameters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

tearth_rng

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

trng_earth72

SOURCE

"ttrcoord.c"

PURPOSE

Performs a linear translation of local range coordinates to earth-centered coordinates (WGS72).

SYNOPSIS

```
#include "ttrcoord.h"

void trng_earth72( E, F, G, x, y, z)
    double *E, *F, *G ;      /* Earth centered coordinates */
    double x, y, z ;         /* local range coordinates */
```

DESCRIPTION

This routine does a linear translation of TTR range coordinates to earth-centered coordinates (Greenwich). Note it does not perform any coordinate rotations. Use 'rotat1' for that function.

It assumes a WGS72 datum for translation parameters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

tearth_rng72

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

wgsnad

SOURCE

"ttrcoord.c"

PURPOSE

Converts WGS84 coordinates to NAD27 coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void wgsnad( wlat, wlong, whgt, hflag, nlat, nlong, nhgt) ;
    double wlat, wlong, whgt ; /* WGS84 lat., long., & height */
    int     hflag ;           /* Height 'msl' flag */
    double *nlat,*nlong,*nhgt; /* NAD27 lat., long., & height
                                (returned) */
```

DESCRIPTION

This routine converts WGS84 coordinates to NAD27 coordinates. It converts the WGS84 latitude, longitude, and height to NAD27 latitude, longitude, and height by using the Molondensky Datum Transformation equations.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

molod

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

xyz

SOURCE

"ttrcoord.c"

PURPOSE

Converts NAD27 geodetic latitude, longitude, and height (msl) into earth-centered coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void xyz( xlat, elong, hght, E, F, G) ;
    double xlat ;      /* geodetic latitude (degrees) */
    double elong ;     /* longitude (degrees) */
    double hght ;      /* Height above Mean Sea Level (MSL) */
    double *E, *F, *G ; /* NAD27 Earth Centered coordinates */
```

DESCRIPTION

This routine takes geodetic latitude, longitude, and height (msl) and converts them into earth-centered coordinates ('E, F, G').

It assumes a NAD27 datum for translation parameters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

rotatl

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

xyz72

SOURCE

"ttrcoord.c"

PURPOSE

Converts WGS72 geodetic latitude, longitude, and height (msl) into earth-centered coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void xyz72( xlat, elong, hght, E, F, G) ;
    double xlat ;      /* geodetic latitude (degrees) */
    double elong ;     /* longitude (degrees) */
    double hght ;      /* Height above Mean Sea Level (MSL) */
    double *E, *F, *G ; /* NAD27 Earth Centered coordinates */
```

DESCRIPTION

This routine takes geodetic latitude, longitude, and height (msl) and converts them into earth-centered coordinates ('E, F, G').

It assumes a WGS72 datum for translation parameters.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

rotatl

SECURITY CLASSIFICATION

UNCLASSIFIED

NAME

xyz_rae

SOURCE

"rngcoord.c"

PURPOSE

Converts TTR range Cartesian coordinates to slant Range, Azimuth, and Elevation coordinates.

SYNOPSIS

```
#include "ttrcoord.h"

void xyz_rae( x, y, z, Rg, Az, El) ;
    double x, y, z ;      /* TTR range Cartesian coordinates */
    double *Rg,*Az,*El ; /* TTR - Range, Azimuth, & Elevation */
```

DESCRIPTION

This routine takes TTR range Cartesian coordinates ('x, y, z') and converts them into Range ('Rg'), Azimuth ('Az'), and Elevation ('El') values.

The Azimuth and Elevation are in units of degrees. With zero(0) degrees referenced to North (Azimuth) and zero(0) degrees horizontal for Elevation.

The 'x, y, z' units are identical to the Range (Rg) units value.

Returns nothing.

AUTHOR

R. H. Meyer, SNL - 2732

SEE ALSO

rae_xyz

APPENDIX A

HP375 Graphics System (hp4) Hardware Specifications

I. Main Chassis [Second from Bottom]

Model# - 375/98588J Computer (S#718737)

1. SPU : MC68030 Processor, 50 Mhz : 13.3 MIPS
MC68882 Floating Point CoProcessor, 50 Mhz
2. Memory : 16 Mbytes, ECC RAM
DMA-CO (HP98620C - equivalent)
3. I/O Interfaces (Integral) : HP98562-66502
 - a. HP-HIL: Keyboard/ Mouse/ ID Module
Select Code = ?
 - b. LAN : Thin-LAN/Ethernet 802.3 (HP98643A - equivalent)
Select Code = 21
 - c. HP-IB : Stand. Speed/Printer & Tape (HP98624A - equivalent)
Select Code = 7
 - d. SCSI S 32 : Disc (HP98265A - equivalent)
Select Code = 14
 - e. RS-232 : Terminal/modem (HP98644A - equivalent)
Select Code = 9
 - f. Parallel : Unused (HP - equivalent)
Select Code = 23
4. High Resolution Video : Color Monitor : HP98550A
Graphics Accelerator : HP98556A
Bit Mapped (1280 x 1024)
Select Code = 132
5. GPIO Interace : HP98622A
16-parallel I/O
Select Code = 12

II. Disk Drive [Bottom]

Model# - 660S (C2213A/306) Mass Storage System (S#718739)

1. Hard Disk : SCSI S32 : Bus Address = 6
Capacity: 663 Mbytes total ; 512 bytes/sector
56 sectors/track ; 1447 tracks/surface
1 head/ surface ; 16 surfaces/ unit

Transfer Rate : 4.0 Mbytes/second (Sync. Burst)
1.5 Mbytes/second (Async. Burst)

2. Optical Disk : Not Installed
3. CD-ROM Disk : Not Installed
4. Audio Tape : Not Installed

III. Tape Drive [Top]

Model# - 9145A Tape Drive (S#718736)

Tape Drive : HP-IB (SC = 7) : HP-IB Address = 3

1/4" Cartridge ; 32 Track

Tape Length - 150 ft. -> 33 Mbytes ; 600 ft. -> 133 Mbytes

Transfer Rate : 4.0 Mbytes/Minute (sustained)

Tape Speed : 120 inches/second

IV. RGB Monitor (19 " High Resolution Color Monitor)(S#718735)

Model# - 98754A Bit Mapped Display

Interface - Device Type : Select Code = 132 : HP98550A (raw - external address space)

Graphics Accelerator : HP98556A

Organization - 2 overlay planes ; 8 Image planes (256 colors)

Size - 15 " x 11.5" (active area - 12.8" x 10.24")

Resolution - 1280 x 1024 pixels (points)

Density - 100 dpi (dots/inch)

V. Input Devices (HP-HIL Devices)

1. Model# 46084A : ID module	: HP-HIL = 1
2. Model# 46021A : Keyboard	: HP-HIL = 2
3. Model# 46060B : Mouse	: HP_HIL = 4
4. Model# 46088B : Graphics Tablet	: HP-HIL = 3

VI. Printer (Color Paintjet)(S#718738)

Model# 3630A : HP-IB (SC = 7) : HP-IB address = 1

Print Rate - 16.7 inches/second
Density - 180 dpi (dots/inch)
Paper size - 8.5" x 11"
Colors - Black, Red, Yellow, Green, Magenta, Blue, Cyan

VII. Graphics Tablet (Serial# 2825A54704)

Model# 46088B : HP-HIL = 3

Size : B-size (11.72" x 17.0" ; 297.6mm x 432.4mm)
Cursor : Fpour Button Crosshair
Resolution : 40k lines per metre
Accuracy : 0.5mm

VIII. Plotter (two pen) (V#935905)

Model# 7470A : HP-IB (SC = 7); HP-IB address = 5

Paper size : A-size (8.5" x 11.0")

Copyright 1989

1406,0

Hewlett-Packard Company

All Rights Reserved

SYSDEBUG

SYSBACKUP

BOOTROM Rev. D

MC68030 Processor

MC68882 Coprocessor

Configuration EEPROM

HP-HIL Keyboard

HP-IB

DMA-CO

RAM 16776964 Bytes

HP98644 (RS232) at 9

HP98622 (GPIO) at 12

HP98265 (SCSI S 32) at 14

HP98643 (LAN) at 21 080009069912

HP Parallel at 23

Bit Mapped Video at 132 (Console)

:HP2213A,

1H SYSHPUX

1D

1B

SEARCHING FOR A SYSTEM (Press RETURN To Pause)

Reset to Power-UP

APPENDIX B

Target Data Block - uVax to HP370 Data and Command

	Data Name	Possible Values	Format	Byte #	Control Source	Note #
1.	GMT Time	milli-second	unsigned long	0 → 3	uVax	1
2.	Radar Targets	position, status, name	array of struct 'Data_Frame'	4 → 195	uVax	2
3.	Reference Point	position, status, name	struct 'Data_Frame'	196 → 219	uVax	3
4.	Course Line	degrees x 10	int	220 → 223	uVax	4
5.	Radar A Index	0 → 7	int	224 → 227	uVax	5
6.	Radar B Index	0 → 7	int	228 → 231	uVax	5
7.	Radar A Display Strings	position, reference, lat, long,	struct 'Data_Strings'	232 → 423	uVax	6
8.	Radar B Display Strings	position, reference, lat, long,	struct 'Data_Strings'	424 → 615	uVax	6
9.	Date String	test date	char 16	616 → 631	uVax	7
10.	Op. Number String	test op. number	char 16	632 → 647	uVax	8
11.	Time-of-Day	Time	char 16	648 → 663	uVax	9
12.	Command	Panel Command	char 32	664 → 695	uVax	10

Note 1: An unsigned (+) 32-bit integer which represents GMT in milliseconds since last midnight.

Note 2: An array of target coordinate data in TTR range coordinates (meters) for each Radar site. The size of the array is determined by the TOTAL_TARGETS macro (8) and each array value is of type 'Data_Frame'.

```
typedef struct {
    int      x ;           /* east of origin */
    int      y ;           /* north of origin */
    int      z ;           /* vertical above origin */
    int      h ;           /* height above MSL */
    Status_des status ;    /* status word */
    char     name[4] ;     /* name of data source */
} Data_Frame ;
```


where the status is defined by the bit-field below.

```
typedef struct {
    unsigned          : 24 ;      /* integer boundary */
    unsigned bit_7      : 1,      /* bit 7 */
    unsigned bit_6      : 1,      /* bit 6 */
    unsigned bit_5      : 1,      /* bit 5 */
    unsigned beacon     : 1,      /* bit 4 */
    unsigned wind       : 1,      /* bit 3 */
    unsigned sonde      : 1,      /* bit 2 */
    unsigned valid      : 1,      /* bit 1 */
    unsigned on_line    : 1,      /* bit 0 */
} Status_des ;
```

Note 3: The current Range reference point in TTR range coordinates (meters) used for all range calculations. The 'Data_Frame' type is the same as is shown in Note 2.

Note 4: The current course line which passes through the Range reference point at the specified angle. The angle is referenced to true North (0 degrees) and the units are in degrees X 10(+ clockwise).

Note 5: These Indices (A & B) are the pointers into the Radar Target array (Note 2) for the currently designated Radar A (B) locations. Radars A & B are the primary (secondary) coordinate locations when performing calculations and plotting.

Note 6: uVax processed string parameters which are displayed in the AlphaNumeric window. They are formatted in the following structure.

```
typedef struct {
    char vel_t[16] ;      /* target total velocity */
    char vel_g[16] ;      /* target ground velocity */
    char x_org[16] ;      /* position - east of TTR origin */
    char y_org[16] ;      /*           - north of TTR origin */
    char z_org[16] ;      /*           - above of TTR origin */
    char x_ref[16] ;      /* position - east of Ref. Point */
    char y_ref[16] ;      /*           - north of Ref. Point */
    char z_ref[16] ;      /*           - above of Ref. Point */
    char h_msl[16] ;      /* height - MSL */
    char h_agl[16] ;      /* height - above ground level */
    char lat[16] ;        /* target latitude - ddd mm ss.t */
    char lon[16] ;        /* target longitude - ddd mm ss.t */
} Data_Strings ;
```

Note 7: uVax processed string test date parameter which is displayed in the AlphaNumeric window. It is in the following form.

Mon-dd-year

Note 8: uVax processed string test operation number parameter which is displayed in the AlphaNumeric window. It is free form with a maximum of 12 characters.

Note 9: uVax processed string time parameter which is displayed in the AlphaNumeric window. It is in the following form.

hh mm ss.t

Note 10: Command string as sent from the Control Panel. The format is given in the following memo.

'Control Panel Switch Functions, Enhanced Graphics System' by D. F. Anderson (2712); August 9, 1991.

Reply Data Block - HP370 to uVax Data

The returned Reply Output Buffer is a fixed length (128 bytes) block, the contents of which are command dependent as follows.

1. Command - Request Operation Number (GQ1) :

Returns : 16 characters

"GQ1 + Operation Number (12 character max.) + NULL(0x00)"

example: "GQ1R12345678901{NULL}"

2. Command - Request 'View' Switch Labels (GQ2) :

Returns : 68 characters

"GQ2 + 8 x {A View Label Names (4 characters)} +
8 x {B View Label Names (4 characters)} + NULL(0x00)"

example: "GQ2AVW0AVW1AVW2AVW3AVW4AVW5AVW6AVW7BVW0BVW
1BVW2BVW3BVW4 BVW5BVW6BVW7{NULL}"

3. Command - Request Time Search Value (A6) :

Returns : 12 characters

"BA6 + {HH:MM:SS} + NULL(0x00)"

example: "BA612:34:56{NULL}"

Appendix B – Target Data Block - uVax to HP370 Data and Command

APPENDIX C

Communication Data Block - Command to Display Procedures

	Data Name	Possible Values	Format	Byte #	Control Source	Note #
1.	Request	String NULL*	char 80	0 → 79	Cmd	A
2.	Test Title	String	char 31	80 → 110	Setup	B
3.	Op. Number	String	char 13	111 → 123	Setup	C
4.	Test Date	String	char 9	124 → 132	Setup	B
5.	Operate Mode	Setup playback Realtime* Timesrch Quit	char 9	133 → 141	Cmd	6
6.	Display Units	Feet* Meters NMiles	char 7	142 → 148	Cmd	7
7.	Trail Length	0,* 10, 100, Max-32767	int	149 → 152	Cmd	8
8.	Peck Rate	10, 1*, .1	float	153 → 160	Cmd	8
9.	Display A View	0 → 8 1*	int	161 → 164	Cmd	5
10.	Display A Segment	1 → 8 all*	int	165 → 168	Cmd	5
11.	Display A xplot	0 → 7	int	169 → 172	Setup	D
12.	Display A yplot	0 → 7	int	173 → 176	Setup	D
13.	Display B View	0 → 8 1*	int	177 → 180	Cmd	5
14.	Display B Segment	1 → 8 all*	int	181 → 184	Cmd	5
15.	Display B xplot	0 → 7	int	185 → 188	Setup	D
16.	Display B yplot	0 → 7	int	189 → 192	Setup	D
17.	Hardcopy	FALSE - 0* TRUE - 1	int	193 → 196	Cmd-Sets GRTP-clears	20
18.	Freeze Screen	FALSE - 0* TRUE - 1	int	197 → 200	Cmd	20
19.	Radar A Assignment	0 → 8 0*	int	201 → 204	Cmd	11
20.	Radar B Assignment	0 → 8 0*	int	205 → 208	Cmd	11
21.	Label	FALSE - 0* TRUE - 1	int	209 → 212	Cmd-sets GRTP-clears	9

Appendix C – Communication Data Block - Command to Display Procedures

	Data Name	Possible Values	Format	Byte #	Control Source	Note #
22.	Label Clear	FALSE - 0 TRUE - 1*	int	213 -> 216	Cmd-clears GRTP-sets	9
23.	Hack	FALSE - 0* TRUE - 1	int	217 -> 220	Cmd-sets GRTP-clears	9
24.	Hack Clear	FALSE - 0 TRUE - 1*	int	221 -> 224	Cmd-clears GRTP-sets	9
25.	Line	CLEAR - 0* MOUSE - 1 KEYBD - 2	int	225 -> 228	Cmd	9
26.	Line Coordinates	Move cord. Draw cord.	float(2) float(2)	229 -> 244 245 -> 260	GRTP	9
27.	AER to X	CLEAR - 0* A TO REF. - 1 REF. TO A - 2 MOUSE TO A - 3	int	261 -> 264	Cmd-sets	9
28.	REF. PT.	1 - 8, 1* KEYBD - 9	int	265 -> 268	Cmd-sets GRTP-clears	10 & 17
29.	IN1/OUT1	FALSE - 0*TRUE - 1	int	269 -> 272	Cmd	16
30.	COORD1	X cord. Y cord. Z cord.	float float float	273 -> 280 281 -> 288 289 -> 296	Cmd	16
31.	IN2/OUT2	FALSE - 0* TRUE - 1	int	297 -> 300	Cmd	16
32.	COORD2	X cord. Y cord. Z cord.	float float float	301 -> 308 309 -> 316 317 -> 324	Cmd	16
33.	SHOW REF. PT.	FALSE - 0* TRUE - 1	int	325 -> 328	Cmd-sets GRTP-clears	17
34.	Plot all Active Radars	FALSE - 0* TRUE - 1	int	329 -> 332	Cmd	E

	Data Name	Possible Values	Format	Byte #	Control Source	Note #
35.	Time Request Response	String	char 80	333 -> 412	GRTP	F
36.	Handshake	FALSE -	int	413 -> 416	Cmd &	G
	Flags(2)	0*TRUE - 1	int	417 -> 420	GRTP	
37.	FORM #	1 -> 5	int	421 -> 424	Setup	H
38.	Window Key Control	0* or 1	int	425 -> 428	GRTP	I
39.	REF. Value	Label, Lat, Long, H, Angle	struct 'reference'	429 -> 489	GRTP	J & 17
40.	Window Coordinates	A*, B*, Alpha*, Cmd, AER, REF	struct 'wndow'	489 -> 533 534 -> 577 578 -> 621 622 -> 665 666 -> 709 710 -> 753	Setup & GRTP	K
41.	Radar plot Parameters (excluding A & B)	Label, marker type, color, size, orientation	array of struct 'radr'	754 -> 1049	Setup	E
42.	View plot Parameters	A: 0 -> 7 B: 0 -> 7	array of struct 'map'	1050 -> 2097 2098 -> 3145	Setup & GRTP	L

* refers to default values which are active at start up.

GRTP - Graphics Real-Time Program

Notes 1 -> 20 refer to notes in the memo 'Control Panel Switch Functions, Enhanced Graphics System' by D. F. Anderson (2712), dated August 9, 1991.

Note 1 Cluster: PLAYBACK FILES

Switch (1,1 -> 1,7) are recognized only in playback mode (Note 6).

PLAYBK 1 -> PLAYBK 4: mutually exclusive switches which select the corresponding log file for playback. Lamp: ON locally when activated, OFF remotely when STOP (this note) is actuated or when eof occurs in the log file.

PAUSE: toggle switch which stops playback throughput from the log file and stops system updates. Pressing PAUSE again resumes throughput.

Lamp: ON locally, OFF locally.

TIME SEARCH: Prompts the graphics operator for time to begin playback throughput from the log file selected by PLAYBK n.

Lamp: ON locally, OFF remotely when time is found.

If time is not found in the log file, display error message at the graphics window and turn off TIME SEARCH and PLAYBK n lamps.

STOP: Stop playback throughput, close the log file, turn off PLAYBK n lamp.

Lamp: ON locally when pressed, OFF remotely when throughput ceases.

Note 2 Cluster: RECORD FILES

Switches (2,1 → 2,7) are recognized in REAL TIME mode (Note 6) only.

RECORD 1 → RECORD 4: mutually exclusive switches which select a file in which to log raw data.

Lamp: ON locally, OFF locally when STOP (this Note) is pressed, or remotely when file size is exceeded.

RECORD RATE 10/SEC, RECORD RATE 1/SEC: mutually exclusive switches which establish recording rate, i.e., 10 frames per second or 1 frame per second.

Lamp: one of the lamps is always ON when in REAL TIME mode (Note 6), otherwise both are OFF.

STOP: Terminates recording and closes the open file.

Lamp: ON locally, OFF remotely when file is closed.

Note 3 Cluster: RADAR STATUS

ON-LINE 1 → ON-LINE 8: Indicator which reflects radar availability. When a radar is not on-line, i.e., not outputting data, the TVTS multiplexer forces alternate 1's and 0's (hex 5555) into each of its four TVTS words. The uVAX real-time program detects this and reports it in the respective radar status byte.

Lamp: ON/OFF remote.

VALID 1 → VALID 8: Indicator which shows radar valid bit status, which is reported in the respective radar status byte.

Lamp: ON/OFF remote.

BCN 1 → BCN 8: Indicator which shows whether or not the radar is using beacon tracking mode. Not implemented because this status bit is not in the current TVTS bit stream.

WIND 1 → WIND 8: Indicator which shows whether or not the radar is tracking a wind balloon. Not implemented because this status bit is not in the current TVTS bit stream.

SOND 1 → SOND 8: Indicator which shows whether or not the radar is tracking a radio sonde. Not implemented because this status bit is not in the current TVTS bit stream.

Note 4 Cluster: PRINT BUFFERS, SAVE BUFFERS

BUFFER 1 (SAVE) → BUFFER 5 (SAVE): mutually exclusive switches which initiate saving Radar A formatted data in the respective memory buffer. The data is stored for 30 seconds at a rate of 1/second and is available for immediate post-test printing when the corresponding BUFFER n (PRINT) is pressed. The Save Buffers are volatile. This function is normally used to save critical trajectory intervals, e.g., apogee of a projectile, when a quick post-test report is required.

Lamp: ON local, OFF remote when 30 samples have been acquired.

BUFFER 1 (PRINT) → BUFFER 5 (PRINT): mutually exclusive switches which initiate printing the contents of the respective SAVE buffer.

Lamp: ON local, OFF remote when printing is complete.

Note 5 Clusters: DISPLAY A SELECT, DISPLAY B SELECT

SEGMENT n: toggle switches that provide on/off control of the eight segments which compose a graphics view (map). The status of the switches will be assembled in two bytes (one byte for A, one byte for B) and sent to the graphics component with each radar data array.

Lamp: ON/OFF local.

VIEW n: mutually exclusive switches within each bank (A and B) which select one of eight maps to be displayed on the graphics screen.

VIEW CLEAR (A) (B): selects no map to be displayed on the graphics screen for (A) or (B).

Note 6 Cluster: OPERATIONAL MODE

REAL TIME, PLAYBK, and SETUP are mutually exclusive. They are used to set the system mode. The system mode determines which switches on the CP are enabled. For example, the RECORD FILES cluster is enabled only in REAL TIME mode and the PLAYBACK FILES cluster is enabled only in PLAYBACK mode.

Lamp: ON/OFF local.

Note 7 Cluster: UNITS SELECT (on main panel)

METERS, FEET, and NAUT MILES are mutually exclusive. They determine the scale factor to be applied to the alphanumeric string data sent to the graphics component. Notice that this has no effect on the binary radar data sent to the graphics system which is always in units of meters relative to the TTR origin.

Lamp: ON/OFF local.

Note 8 Cluster: TRACE CONTROL

TRAIL LENGTH 10, TRAIL LENGTH 100, and TRAIL LENGTH MAX are mutually exclusive. They establish the length of trail to leave on the graphics screen for each radar target. For example, TRAIL LENGTH 10 leaves ten "pen pecks" on the screen for each radar, TRAIL LENGTH 100 leaves 100, and TRAIL LENGTH MAX will be "tuned" when the system is operational.

Lamp: ON/OFF local.

TRAIL CLEAR: clears the buffers containing trail data so that only the current "peck" for each selected radar is displayed.

Lamp: ON local, OFF remote from graphics when erase is completed.

PECK RATE 10/SEC, PECK RATE 1/SEC, PECK RATE 1/10 SEC are mutually exclusive. They set the rate at which the radar samples are "pecked" on the graphics screen. This establishes the density of the trail left by the radar track. Examples: 1/10/SEC would be used when controlling a high speed aircraft near the release point; 1/10 SEC would be used for a balloon track.

Lamp: ON/OFF local.

Note 9 Cluster: GRAPHICS CONTROL

LABEL: Momentary switch which causes a cursor, time, and altitude to be displayed at the track location of Radar A each time the button is pressed. This annotation is in addition to the trail (Note 8) left by the radar track. This feature is used for labeling interesting points along a trajectory, e.g., the apogee of a projectile. Required for hard copy annotations.

LABEL CLEAR: Momentary switch to clear LABELs from the graphics screen.

HACK: Momentary switch which causes a cursor to be displayed at the track position of all selected radars. This cursor is in addition to the trail (Note 8) left by the track.

HACK CLEAR: Momentary switch to clear HACKs from the graphics screen.

AER TO REF PT: Toggle switch which opens a graphics window to display azimuth, elevation, and range alphanumerically; the initial point of the vector is the Radar A track point, the terminal point of the vector is the currently selected reference point (Note 10). This window is maintained until the switch is toggled.

AER FROM REF PT: Same as AER TO REF PT (above) but the vector ends are reversed.

MOUSE REF PT: Same as AER TO REF PT but the initial point of the vector is the current mouse cursor, the terminal point is the Radar A track point.

LINE JOY: Permits the operator to draw line segments on the graphics screen, in real time, that are not part of the permanent or temporary databases. The lines must be maintained between views but are lost when the system goes down. A joy stick is not currently implemented.

LINE KEYBD: Same as LINE JOY but the line endpoints are entered via the keyboard in a graphics window. When this switch is activated the first coordinates entered are a "move," subsequent entries are "draws." When the switch is again actuated the graphics object defined by the lines is closed.

LINE CLEAR: Clears lines from the graphics screen that were entered with LINE JOY or LINE KEYBD.

Note 10 Cluster: REFERENCE POINT SELECT

1 → 8: Mutually exclusive switches that define the point to which all alphanumeric radar data generated by the uVAX real-time program are referenced. Binary coordinate data defining radar track targets sent to the graphics system are not effected. The reference points are currently maintained in a file on the uVAX which is read in at start-up time. Switch annotations are read from the same file.

Note 17 describes the REFERENCE POINT MODIFY cluster which will provide an on-line edit capability to maintain the reference point file.

Lamp: ON/OFF local.

Note 11 Cluster: RADAR SELECT

RADAR A 1 → RADAR A 8: Mutually exclusive switches that provide a selection for the radar to be assigned as Radar A.

RADAR A CLEAR: Clears the Radar A assignment so that Radar A is not displayed on the graphics screen. Use of this switch is rare since Radar A data is the source of many alphanumerically displayed tracking parameters and is also the source for the SAVE and PRINT buffers.

RADAR B 1 → RADAR B 8: Same as above but a selection for Radar B is provided.

RADAR B CLEAR: Clears the Radar B assignment.

Note 12 Cluster: SPECIAL FUNCTION SELECT

1: Monitors communications activity between the CP and the uVAX. If the comm lines are inactive for more than one second the lamp in switch 1 is turned ON as a malfunction warning.

2: Not currently implemented

3: Not currently implemented

4: Not currently implemented

5: Not currently implemented

6: Not currently implemented

7: Not currently implemented

8: Not currently implemented

Note 13 Cluster: UNITS SELECT (Alphanumeric Display Control)

FEET, METERS, NAUT MILES: Mutually exclusive switches to select scaling for the alphanumeric strings displayed on the 6-line LED panel.

Lamp: ON/OFF locally

Note 14 Cluster: LINE SELECT

1 → 8: Selects the line number which is to display the data selected in the next parameter selection (i.e., the next selection from the switches located below the LINE SELECT switches).

Lamp: ON local; OFF when parameter selection is made.

ERASE LINE NUMBER: When activated, the next LINE SELECT selection causes the respective line to be cleared.

Note 15 Cluster: Parameter Display (not labeled on panel)

VEL A: Displays (on the 6-line display) the velocity of the target being tracked by Radar A. This is the vector sum of horizontal velocity and vertical velocity, i.e., total velocity. If UNITS SELECT is feet, the readout is in units of feet/second. If UNITS SELECT is meters, the readout is in units of meters/second. If UNITS SELECT is NAUT MILES, the readout is in units of knots.

VEL B: same as VEL A but source data is from Radar B.

ALT AGL A: The altitude above ground level of the target being tracked by Radar A scaled according to UNITS SELECT.

ALT AGL B: Same as ALT AGL A but source data is from Radar B.

COURSE X DEV A: The perpendicular distance, scaled according to UNITS SELECT, of the target tracked by Radar A from the course line.

COURSE X DEV B: Same as COURSE X DEV A but source data is from Radar B.

ALT MSL A: The altitude above mean sea level of the target tracked by Radar A, scaled according to UNITS SELECT.

ALT MSL B: Same as ALT MSL A but source data is from Radar B.

COURSE Y DEV A: The distance, scaled according to UNITS SELECT, of the target tracked by Radar A from the reference point as determined by REFERENCE POINT SELECT (Note 10).

COURSE Y DEV B: Same as COURSE Y DEV A but source data is from Radar B.

LATA A → LAT B: The latitude of the objects tracked by Radar A and by Radar B respectively.

X ORG A: The distance from the TTR origin X-axis of the target tracked by Radar A.

X ORG B: Same as X ORG A but source data is from Radar B.

LON A → LON B: The longitude of the objects tracked by Radar A and by Radar B respectively.

Y ORG A: The distance from the TTR origin Y-axis of the target tracked by Radar A.

Y ORG B: Same as Y ORG A but source data is from Radar B.

TIME TO GO A: The time-to-go for the target tracked by Radar A to reach the selected reference point (Note 10). This is a down-count based on distance and velocity where a count of zero is the reference point where distance is zero.

TIME OF DAY: Displays the time of day in format HH MM SS.S

Note 16 Cluster 16: SINGLE POINT COORDINATES

IN 1 → IN 2: Initiates the transfer of the contents of the thumbwheel switches to the graphics component where the point they define is plotted as a symbol. Since the thumbwheels are coded in BCD the CP software must translate these to ASCII characters before they are transmitted.

OUT 1 → OUT 2: Sends a signal to the graphics component telling it to remove the symbol plotted by IN n.

Lamp: IN n: ON local, OFF local when OUT n is pressed; the lamp is mutually exclusive with OUT n.

Lamp: OUT n: ON local, OFF local when IN n is pressed; the lamp is mutually exclusive with OUT n.

Note 17 Cluster: REFERENCE POINT MODIFY

KEYBD: Requests the graphics component to open a window that prompts the operator to replace the selected reference point (Note 10) with new coordinates such that the reference point is redefined. The new coordinates are sent to the uVAX as parameters for the radar reduction routine. The coordinates are entered in the form:

Snnnn Lat Lon Hgt Ang S-Switch #

where S is a switch number;

nnnn is a 4-character label that will replace the label in the selected REFERENCE POINT SELECT lcd display;

Lat is the geodetic latitude of the point, DD MM SS.SSS, for example, 37 50 49.396;

Lon is the geodetic longitude of the point in the same format, for example, -116 30 17.489;

Hgt is the altitude of the point in meters relative to mean sea level;

Ang is the angle about which the radar data is to be rotated to generate the COURSE DEV values as discussed in Note 15;

The new reference point should also replace the respective point in the disk file. See Note 10.

SHOW REF PT: Requests the graphics component to open a window to display the coordinates of the selected reference point (Note 10).

Note 18 Cluster: TEST PLAN SELECT

OP TYPE: Not used. Can be re-labeled and used for another function.

START PANEL: Informs the uVAX that the CP processor is running and ready to connect. The uVAX is assumed to be running and waiting for this signal as part of system initialization.

STOP PANEL: Informs the uVAX that the CP is going off-line. This is the first step in an orderly system shutdown.

Note 19 Cluster: ASSIGN VIEW

DISPLAY SELECT A → DISPLAY SELECT B: In SETUP mode (Note 6), assigns the current graphics map (view) to the VIEW NUMBER selected (this Note). In REAL TIME mode (Note 6) that view is selected for display when the respective VIEW is selected (Note 5).

VIEW NUMBER: Works in harmony with DISPLAY SELECT (this Note) to define the map set to support an operation.

Note 20 Cluster: SCREEN CONTROL

FREEZE SCREEN: Recognized in REAL TIME and PLAYBACK modes (Note 6). Requests graphics component to stop updating the screen. Internal processing should continue so that when FREEZE SCREEN is toggled the screen is updated to the current situation, i.e., track trails and alphanumerics are current.

Lamp: ON/OFF local

HARD COPY: Implementation not required because the hard copy device is controlled by its own switch panel.

Note A: The Command process loads this string with either a 'NULL' (empty) string or an ASCII error message. The Display process (GRTP) outputs this string as the first line in the 'Command' window prompt.

Note B: The Setup Graphics process loads this string with a default value from file "STADEF". It is only used locally in the HP370 computer by the Setup process.

Note C: The Setup Graphics process loads this string with a default value from file "STADEF". It is used locally in the HP370 computer by the Setup process and is passed to the MicroVax when Request OpNumber (GQ1) is detected.

Note D: The Setup Graphics process loads these integers with default plot ("WINDEF") values as defined below. They determine what will be plotted on the X & Y axis in the Graphics Real-Time (GRTP) windows (A & B).

```

Target (?) .x      (x_org)  ----- 0
Target (?) .y      (y_org)  ----- 1
Target (?) .z      (z_org)  ----- 2
Radar_S_?.x_ref    (x_ref)  ----- 3
Radar_S_?.y_ref    (y_ref)  ----- 4
Radar_S_?.z_ref    (z_ref)  ----- 5
Target (?) .h      (h_msl)  ----- 6
Radar_S_?.h_agl    (h_agl)  ----- 7

```

Note E: These objects are the eight Radar tracks provided in the 'Target' data block from the MicroVax computer, and are separate plots from the 'A' & 'B' Radar Selects on the control panel. The entire set (all active radars) are turned On/Off by a special function key on the EGS control panel. The Graphics Real-Time Program (GRTP) checks the current 'on_line' bit for each Target to determine which are active Radars. Each plot may have a different marker type, color, size, and orientation. If the 'A' & 'B' are the same radars as any of these plots, they will only be plotted once ('A' or 'B' symbols). The default values are defined in the 'Setup' process and are stored in file "RADDEF".

The eight radars are defined by an array[8] of type struct 'radr' as follows:

```

struct radr {
    char  label[5] ;           /* radar name           */
    int   m_type   ;           /* marker type  1 -> 8   */
    int   m_color  ;           /* marker color index #  */
    float m_size   ;           /* marker size           */
    float m_x, m_y ;           /* marker orientation     */
}

```

Note F: In response to the 'Timesrch' Operate Mode, the Graphics Display process will return operator console entered string. Its form should be as follows, but no check is performed on it by the graphics process.

HH:MM:SS

Note G: Two handshake flags are used to coordinate passage of information between the Command process and the Graphics Real-Time Process (GRTP).

command_flag - the Command process sets this flag to indicate something has changed in Command shared memory. the GRTP clears it and uses it as a display refresh flag.

graphics_flag - the GRTP sets this flag to indicate that it has fulfilled a request from the Command process. the Command clears it when it has read the data.

Note H: The Setup Graphics process loads the Form # with default ("WINDEF") value for AlphaNumeric Window.

Note I: The Graphics Real-Time Process (GRTP) uses the arrow keys, Home key, and Previous/Next keys to Pan/Zoom the A & B Display windows. The current active keyboard window (A = 0 or B = 1) is maintained at this memory location and it is displayed in the AlphaNumeric Window.

Note J: The Reference Point value as entered at the keyboard in response to item#28 (Note 17) above. Values are entered as Latitude, Longitude, and Height and are sent to the MicroVax as a character string.

```
struct reference {
    char label[6]      ;      /* switch# & reference point label      */
    char lat[15]       ;      /* latitude                               */
    char long[15]      ;      /* longitude                              */
    char height[15]    ;      /* height (msl)                           */
    char angle[10]     ;      /* Course Line                           */
}
```

Note K: The windows information is defined for use by the Graphics Real-Time Process (GRTP) for coordination of the viewing surface and any overlaps. The window locations are fixed at 'Setup' time in the setup file ("WINDEF"). Certain windows are fixed in the GRTP as follows:

Graphics: windows A, B, & 'AlphaNumeric' are fixed windows which are continuously displayed during the playback and realtime operational modes.

Command: window 'Cmd' is a popup window used by the command process in playback and realtime operation to query the operator for additional information or display an error message.

Misc: windows 'AER' & 'REF' are temporary use popup windows which are used as required. Coordination is achieved by first checking if the window is currently active.

The windows are defined by a 'wndow' structure as follows:

```
struct wndow {
    float x1, y1      ;      /* rectangle vertices 1 (vdc's)          */
    float x2, y2      ;      /* rectangle vertices 2 (vdc's)          */
    int    type       ;      /* window type: i.e. POPUP,              */
    int    plane      ;      /* image/overlay planes                  */
    int    active     ;      /* window presently active                */
}
```

Note L: The View filenames are associated at 'Setup' time for each View push-button on the control panel. These View files are created by a separate construction program (VIEW) and only the final loading of filenames (with switches) is done by the Graphics Real-Time Program (GRTP). Each window 'A' and 'B' is defined as an array[8] of type 'map' as follows:

```
struct map {  
    int switch ; /* control panel switch # */  
    char view_name[5] ; /* view label name */  
    char path[22] ; /* Path for filename */  
    char filename[16] ; /* filename for map */  
    float xoffset ; /* map zero offset from */  
    float yoffset ; /* range zero ( x, y, z) */  
    float zoffset ;  
    float zoom ; /* map magnification factor */  
    int orientation ; /* map orientation - north */  
    float matrix[3][2] ; /* transformation matrix */  
}
```


APPENDIX D

Starbase/Display List General Map Format

Figure D.1. Graphics Real-Time Program - Simplified Display Loop

PSEUDO CODE

Function		Note
gopen	General Overhead	/* open graphics devices */ 1
set_view_windows		/* set vdc, world, & modeling coordinates; transformations */ 2
set_cull_size		/* set minimum size to display */ 3
set_drawing_mode		/* set replace rule for raster */ 3
set_buffer_mode		/* set single buffered */ 3
set_colors		/* set color_table, background */ 4
set_highlighting		/* set highlighting, color, type, & attributes */ 5
set_marker_attributes		/* set color, size, orientation */ 6
set_text_attributes		/* set text attributes */ 6
set_input_attributes		/* set window, tracking, & size */ 7
set_filter_attributes		/* set invisibility filter */ 8
open_segment(main)	Main View	/* open segment for creation, display, or editing */ 9
matrix_operation		/* magnify, rotate, or translate */ 17
application-data#I		/* segment header information */ 10
add_current_nset		/* add to current nameset */ 11
label(main)		/* main segment label (main) */ 11
view_window(main)		/* set world coordinates */ 17
define element#1		/* Starbase/ Display List primitives */ 12
label(sub1)		/* segment label */ 11
execute segment(sub1)		/* call separate sub-segment */ 13
application-data#II		/* segment(sub1) location info */ 10
define element#2	Segment (sub1)	/* Starbase/ Display List primitives */ 12
label(sub2)		/* segment label */ 11
execute segment(sub2)		/* call separate sub-segment */ 13
application-data#II		/* segment(sub2) location info */ 10
.		
.		
close_segment(main)		/* close segment */ 9
open_segment(sub1)		/* open segment for creation, display, or editing */ 9
application-data#I		/* segment header information */ 10
add_current_nset		/* add to current nameset */ 11
define element#1		/* Starbase/ Display List primitives */ 12
label(subN)		/* segment label */ 11
execute segment(subN)		/* call separate sub-segment */ 13
application-data#II		/* segment(subN) location info */ 10
.		
.		
close_segment(sub1)		/* close segment */ 9
.		
.		

PSEUDO CODE

Function		Note
open_segment(subN)	} Segment (subN) 	/* open segment for creation, display, or editing */ 9
application-data#I		/* segment header information */ 10
add_current_nset		/* add to current nameset */ 11
define element#1		/* Starbase/ Display List primitives */ 12
close_segment(main)		/* close segment from further */ 9 /* access */
refresh_segment(main)		/* draw the map */ 14
make_picture_current		/* force buffer update */ 15
gclose		/* close graphics devices */ 16

1. Before any Starbase command is sent to an I/O device, it must be initialized (opened). Three graphics devices are connected to the HP370 at the present time.

a. display device -- SB_OUTDEV = '/dev/crt' -- image planes
 SB_OVDEV = '/dev/ocrt' -- overlay planes
 SB_OUTDRIVER = 'hp98550'
 SB_OVDRIVER = 'hp98550'

Note: driver = 'hp98556' for graphics accelerator and integer calls.

b. locator/ choice -- SB_INDEV = '/dev/locator'
 device (mouse) SB_INDRIVER = 'hp-hil'

c. keyboard device -- SB_KB = '/dev/tty2'
 SB_KBDRIVER = 'keyboard'

Example Formats:

```
if ((display = gopen(getenv("SB_OUTDEV"), OUTDEV, getenv
("SB_OUTDRIVER", INIT)) == -1)
{ fprintf(stderr, "Error: gopen failed using variables %s and
%s.\n",
getenv("SB_OUTDEV"), getenv("SB_OUTDRIVER") ) ;
exit(-1) ;
}
```



```

if ((overlay = gopen(getenv("SB_OVDEV"), OUTDEV, getenv
    ("SB_OVDRIIVER", INIT)) == -1)
    { fprintf(stderr, "Error: gopen failed using variables %s and
      %s.\n", getenv("SB_OVDEV"), getenv("SB_OVDRIIVER") ) ;
      exit(-1) ;
    }

if ((locator = gopen(getenv("SB_INDEV"), INDEV, getenv
    ("SB_INDRIVER", INIT)) == -1)
    { fprintf(stderr, "Error: gopen failed using variables %s and
      %s.\n", getenv("SB_INDEV"), getenv("SB_INDRIVER") ) ;
      exit(-1) ;
    }

if ((keyboard = gopen(getenv("SB_KB"), INDEV, getenv
    ("SB_KBDRIVER", INIT)) == -1)
    { fprintf(stderr, "Error: gopen failed using variables %s and
      %s.\n", getenv("SB_KB"), getenv("SB_KBDRIVER") ) ;
      exit(-1) ;
    }

```

2. Several viewing transformations must be set up for the Starbase display system. The following table summarizes the EGS values.

	Physical DC	P1, P2	VCD	World Coord. (WC)	Modeling Coord. (MC)
Display	12.8"x10.24" 1280x1024 pixels (0, 0) = upper- left	0.0 → 1.0 Zero = 0.0	Isotropic 0.0 → 1.25 zero = 0.0	Isotropic X, Y meters zero = XO, YO	Isotropic X, Y meters zero = XO, YO
Locator	20.0cmx20.0cm (0, 0) = relative to init position	0.0 → 1.0 zero = 0.0	Isotropic 0.0 → 1.0 zero = 0.0	N/A	N/A

Example Commands Formats for setting windows:

Virtual Device Coordinates (VDC):

```

set_p1_p2(display, FRACTIONAL, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0) ;      (dc)

vdc_extent(display, 0.0, 0.0, 0.0, 1.25, 1.0, 0.0) ;                (vdc)

view_port(display, 0.0, 0.0, 1.0, 1.0) ;                            (vdc)

mapping_mode(display, ISOTROPIC) ;

```

World Coordinates (WC):

view_window(display, -x1, -y1, +x2, +y2) ; (wc) (M)

Modeling Coordinates (MC):

flush_matrices(display) ;

concat_transformation2d(display, xform2, PRE, PUSH) ;

push_matrix2d(display, xform2) ;

pop_matrix(display) ;

Clipping Limits:

clip_rectangle(display, 0.0, 1.0, 0.0, 1.0) ; (vdc)

clip_indicator(display, CLIP_TO_RECT) ;

3. To implement the culling function in the Starbase database the culling limit is set to a diagonal of:

diagonal(DC) = .1 * SQRT(X*X + Y*Y)

diagonal(DC) = .1 * SQRT(1280*1280 + 1024*1024)

diagonal(DC) = 164 pixels (square 116 pixels on a side)

Example Format:

set_cull_size(display, 164) ; (dc)

The replacement rule for raster point operations is set to Source to Destination.

Example Format:

draw_mode(display, 3) ;

The default vertex list format for polygons and polylines is zero extra coordinates, no normals, and clockwise direction.

Example Format:

vertex_format(display, 0, 0, 0, FALSE, CLOCK_WISE) ;

The buffer mode is single buffered with the image planes divided into 1 x 8 planes and the 2 overlay planes transparent.

Example Format:

```
double_buffer(display, DbufferOff, 4) ;

dbuffer_switch(display, !buffer) ;

buffer_mode(display, On) ;

flush_buffer(display) ;

display_enable(display, 0xF << (4*BufSwh) ) ;

write_enable(display, 0xF << (4*!BufSwh) ) ;

gescape(overlay, GR2D_OVERLAY_TRANSPARENT, &arg1, &arg2) ;
```

4. The EGS system monitor is divided into 2 overlay planes and 8 image planes. Thus the number of available colors in the image planes is 256 colors, while the number of colors in the overlay planes is limited to 4 colors.

DEFAULT 16-Entry Color Map

Index	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	white	1.0	1.0	0.0
2	red	1.0	0.0	1.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0
8	10% gray	0.1	0.1	0.1
9	20% gray	0.2	0.2	0.2
10	30% gray	0.3	0.3	0.3
11	40% gray	0.4	0.4	0.4
12	50% gray	0.5	0.5	0.5
13	60% gray	0.6	0.6	0.6
14	70% gray	0.7	0.7	0.7
15	80% gray	0.8	0.8	0.8

The background color is black (0).

Example Command Formats:

```

inquire_sizes(display, phy_limits, resolution, p1, p2, cmap_size) ;

inquire_color_table(display, 0, cmap_size, color_table) ;

gescape(display, READ_COLOR_MAP, &arg1, &arg2) ;

define_color_table(display, 0, cmap_size, color_table) ;

background_color(display, 0.0, 0.0, 0.0) ; (M)
background_color_index(display, 0) ; (M)

clear_control(display, CLEAR_VIEWPORT) ;

clear_view_surface(display) ;

```

5. The highlighting function is used to emphasize certain objects (elements or segments). This is mainly used with the 'picking' function to denote selected objects.

Example Command Formats:

```

highlight_on(display, Off) ;

highlight_color(display, 1.0, 1.0, 1.0) ;
highlight_color_index(display, 1) ;

highlight_type(display, SOLID) ;

highlight_attributes(display, HL_COLOR | HL_STYLE) ;

```

6. The markers are used as symbols to plot each radar track. At the current time 18 markers are available in the marker fonts library. Markers are created the same way text fonts are created.

Example Command Formats:

```

marker_color(display, 0.5, 0.5, 0.5) ; (M)

marker_color_index(display, 12) ; (M)

marker_size(display, 0.05, FALSE) ; (vdc) (M)

dcmarker_size(display, 50) ; (dc)

marker_orientation(display, 0.0, 1.0) ; (vdc) (M)

```

The text capability is used to annotate all selected points on the display views.

Example Command Formats:

```

text_precision(display, STROKE_TEXT) ;

```

6. Continued.

```

designate_character_set(display, "usascii") ;           (M)

text_font_index(display, 1) ;                         (M)

character_height(display, 0.05) ;                     (vdc) (M)
dccharacter_height(display, 50) ;                     (dc)

character_width(display, 0.036) ;                     (vdc) (M)

text_alignment(display, TA_LEFT, TA_HALF, 0.0, 0.0) ; (wc) (M)

text_path(display, PATH_RIGHT) ;                      (M)

text_line_path(display, PATH_DOWN) ;                  (M)

text_orientation2d(display, 0.0, 1.0, 1.0, 0.0) ;     (vdc) (M)

```

7. The tracking function uses the mouse (locator) in the Real-Time Program (GRTP) to select display points (AER to X) or draw a line (LINE) in window A.

Example Command Formats:

```

track_off(locator) ;

track(locator, display, 1) ;

echo_type(display, 0, SMALL_TRACKING_CROSS, 0.5, 0.5, 0.0) ; (vdc)

echo_update(display, 0, 0.5, 0.5, 0.0) ;              (vdc)

set_locator(locator, 1, 0.5, 0.5, 0.0) ;             (vdc)

enable_events(locator, LOCATOR, 1) ;

await_event(locator, .05, &valid, &class) ;

read_locator_event( -1, &locator_fildes, &ordinal, &x, &y, &z, (vdc)
&status, &message_link) ;

disable_events(locator, ALL, 1) ;

```

The keyboard 'choice' function gets keystroke values and processes them in the reference (REF. PT.) input mode or the line (LINE) draw mode.

Pop-up windows are used.

Example Command Formats:

```

enable_events(keyboard, CHOICE, 1) ;

```

7. Continued.

```

await_event(keyboard, .05, &valid, &class) ;

read_choice_event( -1, &choice_fildes, &ordinal, &value, &status,
    &message_link) ;

disable_events(Keyboard, ALL, 1) ;

block_read(display,x_left,y_up,x_len,y_len,&Save_area,FALSE) ;
    (vdc/dc)
dcblock_read(display,x_left,y_up,x_len,y_len,&Save_area,FALSE) ;
    (dc)

block_write(display,x_left,y_up,x_len,y_len,&Save_area,FALSE) ;
    (vdc/dc)
dcblock_write(display,x_left,y_up,x_len,y_len,&Save_area,FALSE) ;
    (dc)

```

The pick function is not used in the Real-Time Program (GRTP). It allows the operator to select an object from the screen. At the present time picking is limited to individual elements (not Display List).

Example Command Formats:

```

set_pick_window(display, x_left, y_low, x_right, y_up) ;      (vdc)

set_hit_mode(display, FALSE) ;

inquire_hit(display, &hit) ;

```

8. The invisibility filter is used in the EGS Graphics Real-Time Program (GRTP) to selectively turn segments on/off with the Control Panel 'Segment' switches.

Example Command Formats:

```

set_invisibility_filter(display, 32, InvisIncl, 32, InvisExcl) ;

```

9. Each view in the EGS graphics database consists of a main segment and other segments. The main segment may contain individual elements, but will also contain calls to other segments. Each of these called segments may be selectively opened, filtered, edited, and closed.

As such, each segment is assigned a segment number, a label number, and a nameset value. The three numbers are set to identical values in the EGS graphics database (see also note 11).

Segment numbers are integers that are uniquely associated with each segment (at load time). They are selected according to the following criteria.

- A. The Main Segments (View) correspond to the Control Panel 'View' switches.

1, 2, 3, 4, 5, 6, 7, 8

- B. Segments directly called by a Main Segment correspond to the 'Segment' switches on the Control Panel (if they are filterable).

i.e. Main Segment(1) -> Segment(1x)

11, 12, 13, 14, 15, 16, 17, 18

- C. Segments indirectly called by a Main Segment via another Segment (see B.).

i.e. Main Segment(1) -> Segment(1x) -> Segment(1xx)

111, 112, 113, 114, 115, 116, 117, 118

Each segment must be 'opened' before it can be accessed. Only one segment per Display List may be open at a time. The segment should be closed when finished.

Example Command Formats:

```
open_segment (display, segno, AppendOff, DisplayOff) ;
```

```
close_segment (display) ;
```

10. Application data is the Display List mechanism by which small amounts of general data may be stored with the graphics data. In the EGS database it consists of two general types (I & II) and is used to store housekeeping functions for constructing the 'Views'.

- A. Type application-data#I includes all data which is pertinent to a segment in a separate file. It is used to identify separate entities in the Display List. It will always be the first element in any segment.

```
Type #I: struct AppData {
            char Remark[81] ;      /* Remarks & Comments */
            char Author[21] ;      /* Author */
            char Date[11] ;        /* Date - Last revised */
            int Rev ;               /* Current revision number */
            struct map mp_data;    /* map structure */
        }

        struct map {
            int switch ;           /* Control Panel switch # */
            char view_name[5] ;    /* Control View Name */
            struct file_seg file;  /* this filename/ location */
            float xoffset ;        /* map zero offset from... */
            float yoffset ;        /* range zero (x, y, z) */
            float zoffset ;
            float zoom ;           /* map magnification factor */
            int orientation ;      /* map azimuth - north */
        }
```

- B. Type application-data#II contains the name and location of a file segment. It is used to locate an external Display List segment in a disk file. It must always follow a "call_segment/execute_segment" command.

```

Type #II: struct file_seg {
            char path[22] ;      /* Path for filename */
            char filename[16] ; /* filename for segment */
        }

```

Example Command Formats:

```

application_data(display, sizeof(AppData), &AppData) ;

inq_application_data_size(display, &size) ;

inq_application_data(display, &AppData) ;

```

11. Each view in the EGS graphics database consists of a main segment and other segments. The main segment may contain individual elements, but will also contain calls to other segments. Each of these called segments may be selectively opened, filtered, edited, and closed.

As such, each segment is assigned a segment number, a label number, and a nameset value. The three numbers are set to identical values in the EGS graphics database (see also note 9).

EGS label numbers are integers which identify discrete portions of segments. In the main segment, the main label identifies the start of the displayable graphics elements. Segment labels identify the point in a calling segment where the call to another segment is initiated.

Label numbers and segment numbers have different functions. Segment numbers identify the segment which is being called, while segment labels identify where it was called from. Label numbers are used mostly in the editor and in setup mode.

Example Command Format:

```

dl_label(display, Seg1) ;

```

Namesets are used in the Display List filtering process. They are an array of integers whose individual bits are compared to determine if the list is tranversed or not tranversed. The nameset numbers identify bit locations in the 'current' set and in the 'invisibility' filter set. When a segment is first tranversed the number is entered into the active 'current' set. If this 'current' set value (bit location) is identical to the 'invisibility' set value, then this segment is skipped (Invisible - not tranversed).

The 'invisibility' set is changed according to the settings of the Control Panel 'Segment' switches (see Note 8) .

The relationship between the segment numbers, label numbers, and nameset numbers is shown below.

Assume Main Segment (View) number = x ;

		<u>Seg#</u>	<u>Label#</u>	<u>Nameset#</u>
#define	Seg1	x1	x1	x1
#define	Seg2	x2	x2	x2
#define	Seg3	x3	x3	x3
#define	Seg4	x4	x4	x4
#define	Seg5	x5	x5	x5
#define	Seg6	x6	x6	x6
#define	Seg7	x7	x7	x7
#define	Seg8	x8	x8	x8

Note: Any segments called by the above segments use the next available numbers in the sequence.

i.e. x11, x12, etc.

Example Command Formats:

```
remove_all_names_from_set(display) ;
add_names_to_set(display, 32, Nameset) ;
remove_names_from_set(display, 32, Nameset) ;
inq_nameset_size(display, &size) ;
inq_nameset(display, &Nameset) ;
```

12. The body of the EGS segment consists of drawing primitives which comprise the main 'View' and 'Segments'. Example formats and topics follow.

Example Drawing File Formats:

Line Routines

```
move2d(display, x, y) ; (wc) (M)
draw2d(display, x, y) ; (wc) (M)
polyline2d(display, clist[], numpts, TRUE) ; (wc) (M)
line_color(display, red, green, blue) ; (M)
line_color_index(display, 4) ; (M)
line_endpoint(display, SQUARE) ; (M)
line_repeat_length(display, 0.039) ; (vdc) (M)
line_type(display, SOLID) ; (M)
```

line_width(display, 0,0, VDC_UNITS) ; (vdc/mc) (M)

Marker Routines

polymarker2d(display, clist[], numpts, TRUE) ; (wc) (M)

marker_type(display, 2) ; (M)

marker_color(display, 0.5, 0.5, 0.5) ; (M)

marker_color_index(display, 12) ; (M)

marker_size(display, 0.05, FALSE) ; (vdc) (M)

marker_orientation(display, 0.0, 1.0) ; (vdc) (M)

Polygon Routines

polygon2d(display, clist[], numpts, FALSE) ; (wc) (M)

partial_polygon2d(display clist[], numpts, FALSE, TRUE) ; (wc) (M)

rectangle(display, xmin, ymin, xmax, ymax) ; (wc) (M)

vertex_format(display, 0, 0, 0, FALSE, CLOCK_WISE) ; (M)

fill_color(display, red, green, blue) ; (M)

fill_color_index(display, 4) ; (M)

hatch_orientation(display, 1.0, 0.0) ; (M)

hatch_spacing(display, 0.039, VDC_UNITS) ; (dc/vdc/mc) (M)

hatch_type(display, PARALLEL_HATCH) ; (M)

interior_style(display, INT_SOLID, NoEdge) ; (M)

perimeter_color(display, red, green, blue) ;

perimeter_color_index(display, 4)

perimeter_repeat_length(display, 0.039) ; (vdc)

perimeter_type(display, SOLID) ;

Curve Routines

arc(display, x_rad, y_rad, x_center, y_center, start, (wc) (M)
stop, rotation, NO_CHORD) ; (radians)

partial_arc(display, x_rad, y_rad, x_center, y_center, start, (wc) (M)
stop, rotation, NO_CHORD, FALSE) ; (radians)

ellipse(display, x_rad, y_rad, x_center, y_center, (wc) (M)
rotation) ; (radians)

partial_ellipse(display, x_rad, y_rad, x_center, y_center (wc) (M)
rotation, FALSE) ; (radians)

interior_style(display, INT_HALLOW, Edged) ; (M)

curve_resolution(display, STEP_SIZE, 1.0 DEG, 1.0 DEG, (M)
1.0 DEG, 1.0 DEG) ; (radians)

Text Routines

text2d(display, x, y, string, VDC_TEXT, EndOfLine) ; (vdc/wc) (M)

appendtext(display, string, VDC_TEXT, EndOfLine) ; (vdc/wc) (M)

character_expansion_factor(display, 1.0) ; (M)

character_height(display, 0.05) ; (vdc) (M)

character_width(display, 0.036) ; (vdc) (M)

character_slant(display,, 0,0) ; (M)

designate_character_set(display, "usacii") ; (M)

text_alignment(display, TA_LEFT, TA_HALF, 0.0, 0.0) ; (wc) (M)

text_color(display, red, green, blue) ; (M)

text_color_index(display, 4) ; (M)

text_font_index(display , 1) ; (M)

text_line_path(display, PATH_DOWN) ; (M)

text_orientation2d(display, 0.0, 1.0, 1.0, 0.0) ; (vdc) (M)

text_path(display, PATH_RIGHT) ; (M)

Miscellaneous

background_color(display, 0.0, 0.0, 0.0) ; (M)

background_color_index(display, 0) ; (M)

view_window(display, -x1, -y1, +x2, +y2) ; (wc) (M)

13. Each segment in the Display List may call other segments in the same Display List. There are several methods of calling another segment.

The following command calls a segment without saving the current Starbase state.

Example Command Format:

call_segment(display, segno) ;

The following commands allow the current Starbase state to be saved before the call and restored upon return.

Example Command Formats:

```
push_state(display) ;
pop_state(display) ;
execute_segment(display, segno) ;
```

In addition, segments may be conditionally called (Pruning & Culling) if certain conditions are met (with/without saving state).

Example Command Formats:

```
cond_call_segment(display, CI_PRUNE | CI_CULL, FALSE, segno) ;
cond_execute_segment(display, CI_PRUNE | CI_CULL, FALSE, segno) ;
cond_return(display, CI_PRUNE | CI_CULL, TRUE) ;
set_extent(display, MExtent[]) ;
```

14. After the Starbase Display List is setup, it must be output to the screen (device). This is assuming that the 'open_segment' DisplayOff parameter was set or it is desired to refresh the screen without re-executing the commands explicitly.

Example Command Formats:

```
refresh_segment(display, segno) ;
refresh_element(display) ;
```

15. Since the EGS graphics program is buffered, the program must make sure all primitives have been flushed to the screen. This can be performed with or without waiting for completion.

Example Command Formats:

```
flush_buffer(display) ;
make_picture_current(display) ;
```

16. After all Starbase commands have been executed on the I/O device, the device should be disconnected (closed).

Example Command Formats:

```
gclose(display) ;
gclose(overlay) ;
gclose(locator) ;
gclose(keyboard) ;
```

17. An overall 'view_window' (meters) is assigned for each main segment. This sets the window size in world coordinates for all sub-segments. Any sub-segment window is converted to this 'view_window' at build time.

Example Command Formats:

```
view_window(display, -x1, -y1, +x2, +y2) ; (wc) (M)
```

```
flush_matrices(display) ;
```

```
concat_transformation2d(display, mat, POST, REPLACE) ;
```

```
push_matrix2d(display, mat) ;
```

```
pop_matrix(display) ;
```

```
pop_matrix2d(display, mat) ;
```

APPENDIX E
EGS Database File Formats

Figure E.1. Building EGS Segments and Views

The map files for the EGS Graphics system may be stored in any of four file formats.

ASCII File Format

The 'ASCII' file format (*.asc) is a direct text representation of the Starbase/ Display List graphics commands. It is the simplest format for user to read and interrupt. Also, it is the most useful when transferring drawings between graphics systems or when building EGS maps in a nonStarbase application ('vi', TCALL, WAYPOINTS).

One subtlety in using this 'ASCII' format is with the 'application_data' Display List command. It is a byte oriented structure while most text editors ('vi') are word oriented. This fact can cause unexpected results when a text editor is used to change a line in the 'application_data' command. Therefore, it is suggested that the library subroutines 'read_asc_in', 'write_asc_out', and 'Asc_copy' be used when manipulating these files. Another method would be to use the default 'header.asc' and then edit it with the 'SHOW_SEG' program. This procedure is further explained in Appendix H.

Segment File Format

The Display List segment format (*.sfl) is analogous to a 'compiled' version of the ASCII (*.asc) file format. It can be quickly displayed (traversed) when loaded into another Starbase application program. The segment format is a binary (integer) file which is defined in the Display List library and can be edited using Display List commands. However, it is not easily read by a user.

The library subroutines 'write_display_list' and 'read_display_list' may be used to save/restore these files. The library subroutine 'PrintSegment' can be used to 'reverse compile' the segment file and printout the results in a form which is more easily readable.

Program SHOW_SEG (ConvMap) will read an 'ASCII' file (*.asc) and produce a segment file (*.sfl) as an output.

Program PrtDLst will read a segment file (*.sfl) and print a text listing, but it is not directly in the 'ASCII' file (*.asc) format.

View File Format

The Display List view format (*.vfl) is identical to the segment format with the following changes.

1. The 'application_data' header information is complete.
2. The Starbase filter 'NameSet' header is configured in the main and sub-segments.
3. The EGS hierarchy is fully implemented.
i.e. All Segment files (*.sfl) are loaded and callable from the main segment.
4. A main 'View_Window' is defined for the view world coordinate system.

Program BUILD (ViewBld) creates the view file (*.vfl) from either 'ASCII' files (*.asc) or segment files (*.sfl).

The same library subroutines 'write_display_list' and 'read_display_list' are used to save/restore the view files. Also, the program PrtDLst can be used to print a text listing of a view file (*.vfl).

Bitmap Image Format

The fourth format is the 'bitmap' format which is used to store a frame buffer or memory image of the screen. This format is used by the EGS graphics realtime program to store screen views for subsequent 'spooled' output to the printer. It is not used, at this time, for any other purpose.

I. ASCII File Format

A. example: nellis.asc

```

application_data(3, 207,data)
File converted from TTR Tektronix (*.com) file.
Unknown          2-10-1992   1   0   /users/meyer/ttrmap
nellis.sfl        0         0   0   1   0 ;
remove_all_names_from_set( 3);
add_names_to_set( 3, 32, NameSet)
      2,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0;

dl_label( 3, 1);
view_window( 3, -38000.0,-200000.0, 275000.0, 113000.0);
application_data( 3, 20, data)
NELLIS.COM 21MAR84;
application_data( 3, 32, data)
AFR 50-46/NAFB SUP 1 28 AUG 1983;
polyline2d( 3, clist, 4, 1)
  204247.0, 47715.0, 0,
  204647.0, 47715.0, 1,
  204447.0, 47515.0, 0,
  204447.0, 47915.0, 1;
text2d( 3, 204447.0, 47915.0, ILC,3,0);
polyline2d( 3, clist, 4, 1)
  -26854.0, 20838.0, 0,
  -26454.0, 20838.0, 1,
  -26654.0, 20638.0, 0,
  -26654.0, 21038.0, 1;
text2d( 3, -26654.0, 21038.0, TPH,3,0);
polyline2d( 3, clist, 4, 1)
  141225.0,-194540.0, 0,
  141625.0,-194540.0, 1,
  141425.0,-194740.0, 0,
  141425.0,-194340.0, 1;
text2d( 3, 141425.0,-194340.0, LAS,3,0);
polyline2d( 3, clist, 4, 1)

```

```

218698.0,-116330.0, 0,
219098.0,-116330.0, 1,
218898.0,-116530.0, 0,
218898.0,-116130.0, 1;
text2d( 3, 218898.0,-116130.0, MMM,3,0);
polyline2d( 3, clist, 4, 1)
-1778.0,-115763.0, 0, -1378.0,-115763.0, 1,
-1578.0,-115963.0, 0,
-1578.0,-115563.0, 1;
text2d( 3, -1578.0,-115563.0, BTY,3,0);
move2d( 3, 31842.0,-137241.0);
draw2d( 3, 13807.0, -60822.0);
move2d( 3, 12278.0, -53689.0);
draw2d( 3, -4075.0, 17228.0);
polyline2d( 3, clist, 4, 1)
23364.0, 93302.0, 0,
23764.0, 93302.0, 1,
23564.0, 93102.0, 0,
23564.0, 93502.0, 1;
text2d( 3, 23564.0, 93502.0, CP,3,0);

```

B. example: view4.asc

```

application_data(3, 207,data)
'View' file - built with 'ViewBld' program.
Unknown          4- 7-1992   1   4 VWA4 /users/egs/data
view4.vf1   0           0           0           1   0 ;
remove_all_names_from_set( 3);
add_names_to_set( 3, 32,NameSet)
      10,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0,          0,          0,          0,          0,
      0,          0;
dl_label( 3, 4);
view_window( 3, -40000.0, -40000.0, 40000.0, 40000.0);
dl_label( 3,41);
execute_segment( 3, 41);
application_data(3, 37,data)

```

```

/users/egs/data      nellis.asc      ;
dl_label( 3,42);
execute_segment( 3, 42);
application_data(3, 37,data)
/users/egs/data      nlabel.asc      ;
dl_label( 3,43);
execute_segment( 3, 43);
application_data(3, 37,data)
/users/egs/data      nrange.asc      ;
dl_label( 3,44);
execute_segment( 3, 44);
application_data(3, 37,data)
/users/egs/data      roads.asc       ;
dl_label( 3,45);
execute_segment( 3, 45);
application_data(3, 37,data)
/users/egs/data      lakesa.asc      ;
dl_label( 3,46);
execute_segment( 3, 46);
application_data(3, 37,data)
/users/egs/data      lakesn.asc      ;
dl_label( 3,47);
execute_segment( 3, 47);
application_data(3, 37,data)
/users/egs/data      radar.asc       ;
dl_label( 3,48);
execute_segment( 3, 48);
application_data(3, 37,data)
/users/egs/data      mancmp.asc      ;

```

II. Display List 'Segment' & 'View' File Formats

A. Format on Disk (Integers)

Note: This is not valid 'C' code.

1. Element Structure (variable size)

```

struct element {
    int element_size ;           /* # of members that follow */
    int member[element_size] ;  /* members of element */

```

```
} ele#? ;
```

2. Segment Structure (variable size)

```
struct segment {
    int segment_number ;           /* segment number */
    struct element ele#0 ;         /* first element */
    struct element ele#1 ;         /* second element */
    .
    .
    .
    struct element ele#N ;         /* N th element */
    int zero = 0 ;                 /* zero size element */
} seg#? ;
```

3. View File Structure (variable size)

```
struct segment seg#0 ;           /* first (main) segment */
struct segment seg#1 ;           /* second segment */
.
.
.
struct segment seg#M ;           /* M th segment */
EOF                               /* End-Of-File */
```

E. example: PrtDLst nellis.sfl

Segment # - 1

ELEMENT 0

application_data(fd, 207,

```
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472
6d617020 20206e65 6c6c6973 2e73666c 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })
```

remove_all_names_from_set(fd)

add_names_to_set(fd, 32,

```
{
    2,      0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,      0,
```

Appendix E – EGS Database File Formats

```

0,      0,      0,      0,      0,      0,
0,      0}))
dl_label( fd, 1)
view_window( fd, -38000, -200000, 275000, 113000)
application_data( fd, 20,
{4e454c4c 49532e43 4f4d2020 2032314d 41523834 })
application_data( fd, 32,
{41465220 35302d34 362f4e41 46422053 55502031 20323820 41554720
31393833 })
polyline2d( fd,
{ 204247, 47715, 0,
204647, 47715, 1,
204447, 47515, 0,
204447, 47915, 1}, 4, 1)
text2d( fd, 204447, 47915,"ILC", 3, 0)
polyline2d( fd,
{ -26854, 20838, 0,
-26454, 20838, 1,
-26654, 20638, 0,
-26654, 21038, 1}, 4, 1)
text2d( fd, -26654, 21038,"TPH", 3, 0)
polyline2d( fd,
{ 141225, -194540, 0,
141625, -194540, 1,
141425, -194740, 0,
141425, -194340, 1}, 4, 1)
text2d( fd, 141425, -194340,"LAS", 3, 0)
polyline2d( fd,
{ 218698, -116330, 0,
219098, -116330, 1,
218898, -116530, 0,
218898, -116130, 1}, 4, 1)
text2d( fd, 218898, -116130,"MMM", 3, 0)
polyline2d( fd,
{ -1778, -115763, 0,
-1378, -115763, 1,
-1578, -115963, 0,
-1578, -115563, 1}, 4, 1)
text2d( fd, -1578, -115563,"BTY", 3, 0)
move2d( fd, 31842, -137241)

```

```

draw2d( fd, 13807, -60822)
move2d( fd, 12278, -53689)
draw2d( fd, -4075, 17228)
polyline2d( fd,
{
    23364,      93302,      0,
    23764,      93302,      1,
    23564,      93102,      0,
    23564,      93502,      1}, 4, 1)
text2d( fd, 23564, 93502, "CP", 3, 0)

```

F. example: PrtDLst view4.vfl

```

Segment # - 4
ELEMENT 0
application_data( fd, 207,
{27566965 77272066 696c6520 2d206275 696c7420 77697468 20275669 6577426c 64272070
726f6772 616d2e20 20202020 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202034 2d20372d
31393932 20202031 20202020 34202056 57413420 2f757365 72732f65 67732f64 61746120
20202020 20207669 6577342e 76666c20 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })
remove_all_names_from_set( fd)
add_names_to_set( fd, 32,
{
    16,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0})
dl_label( fd, 4)
view_window( fd, -40000, -40000, 40000, 40000)
dl_label( fd, 41)
execute_segment( fd, 41)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20206e65 6c6c6973 2e617363 20202020
20  })
dl_label( fd, 42)
execute_segment( fd, 42)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20206e6c 6162656c 2e617363 20202020
20  })
dl_label( fd, 43)

```


Appendix E – EGS Database File Formats

```
execute_segment( fd, 43)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20206e72 616e6765 2e617363 20202020
20      })
dl_label( fd, 44)
execute_segment( fd, 44)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 2020726f 6164732e 61736320 20202020
20      })
dl_label( fd, 45)
execute_segment( fd, 45)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20206c61 6b657361 2e617363
20202020 20      })
dl_label( fd, 46)
execute_segment( fd, 46)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20206c61 6b65736e 2e617363 20202020
20      })
dl_label( fd, 47)
execute_segment( fd, 47)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20207261 6461722e 61736320 20202020
20      })
dl_label( fd, 48)
execute_segment( fd, 48)
application_data( fd, 37,
{2f757365 72732f65 67732f64 61746120 20202020 20206d61 6e636d70 2e617363 20202020
20      })
```

Segment # - 41

element 0

```
application_data( fd, 207,
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472
6d617020 20206e65 6c6c6973 2e73666c 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020      })
remove_all_names_from_set( fd)
add_names_to_set( fd, 32,
{      0,      512,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
```

```

0,      0,      0,      0,      0,      0,
0,      0,      0,      0,      0,      0,
0,      0,      0,      0,      0,      0,
0,      0}))

application_data( fd, 20,
{4e454c4c 49532e43 4f4d2020 2032314d 41523834 }) application_data( fd, 32,
{41465220 35302d34 362f4e41 46422053 55502031 20323820 41554720 31393833 })
polyline2d( fd,
{      204247,      47715,      0,
      204647,      47715,      1,
      204447,      47515,      0,
      204447,      47915,      1}, 4, 1)
text2d( fd, 204447, 47915,"ILC", 3, 0)
polyline2d( fd,
{      -26854,      20838,      0,
      -26454,      20838,      1,
      -26654,      20638,      0,
      -26654,      21038,      1}, 4, 1)
text2d( fd, -26654, 21038,"TPH", 3, 0)
polyline2d( fd,
{      141225,      -194540,      0,
      141625,      -194540,      1,
      141425,      -194740,      0,
      141425,      -194340,      1}, 4, 1)
text2d( fd, 141425, -194340,"LAS", 3, 0)
polyline2d( fd,
{      218698,      -116330,      0,
      219098,      -116330,      1,
      218898,      -116530,      0,
      218898,      -116130,      1}, 4, 1)
text2d( fd, 218898, -116130,"MMM", 3, 0)
polyline2d( fd,
{      -1778,      -115763,      0,
      -1378,      -115763,      1,
      -1578,      -115963,      0,
      -1578,      -115563,      1}, 4, 1)
text2d( fd, -1578, -115563,"BTY", 3, 0)
move2d( fd, 31842, -137241)
draw2d( fd, 13807, -60822)
move2d( fd, 12278, -53689)

```

Appendix E – EGS Database File Formats

```

draw2d( fd, -4075, 17228)
polyline2d( fd,
{
    23364,    93302,    0,
    23764,    93302,    1,
    23564,    93102,    0,
    23564,    93502,    1}, 4, 1)
text2d( fd, 23564, 93502,"CP", 3, 0)

    Segment # - 42
ELEMENT 0
application_data( fd, 207,
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 204c2e20 4c2e2059 6f756e67 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472
6d617020 20206e6c 6162656c 2e73666c 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })
remove_all_names_from_set( fd)
add_names_to_set( fd, 32,
{
    0,          1024,    0,          0,          0,          0,
    0,          0,      0,          0,          0,          0,
    0,          0,      0,          0,          0,          0,
    0,          0,      0,          0,          0,          0,
    0,          0,      0,          0,          0,          0,
    0,          0})
application_data( fd, 23,
{4c414245 4c532046 4f52204e 454c4c49 53205241 4e4745  })
application_data( fd, 25,
{20555044 41544544 20303520 4a414e20 38392c20 20204c4c 59  })
text2d( fd, 91500, 39500,"REVEILLE", 3, 0)
text2d( fd, 101700, -26500,"COYOTE", 3, 0)
text2d( fd, 93500, -6200,"CHARLIE", 3, 0)
text2d( fd, 127000, -44500,"BRAVO", 3, 0)
text2d( fd, 93500, -38500,"ALPHA", 3, 0)
text2d( fd, 178000, -15500,"CALIENTE", 3, 0)
text2d( fd, 170160, -3300,"WEST", 3, 0)
text2d( fd, 210000, -27500,"EAST", 3, 0)
text2d( fd, 155000, -10000,"A", 3, 0)
text2d( fd, 155000, -15000,"L", 3, 0)
text2d( fd, 155000, -20000,"P", 3, 0)
text2d( fd, 155000, -25000,"H", 3, 0)
text2d( fd, 155000, -30000,"A", 3, 0)

```

```
text2d( fd, 182000, -71500, "ELGIN", 3, 0)
text2d( fd, 173000, -62500, "NORTH", 3, 0)
text2d( fd, 193500, -81500, "SOUTH", 3, 0)
text2d( fd, 133000, -80000, "A", 3, 0)
text2d( fd, 133000, -86000, "L", 3, 0)
text2d( fd, 133000, -92000, "A", 3, 0)
text2d( fd, 133000, -98000, "M", 3, 0)
text2d( fd, 133000, -104000, "O", 3, 0)
text2d( fd, 130500, -68000, "N", 3, 0)
text2d( fd, 130500, -119000, "S", 3, 0)
text2d( fd, 148000, -78000, "S", 3, 0)
text2d( fd, 148000, -84000, "A", 3, 0)
text2d( fd, 148000, -90000, "L", 3, 0)
text2d( fd, 148000, -96000, "L", 3, 0)
text2d( fd, 148000, -102000, "Y", 3, 0)
text2d( fd, 155000, -70000, "C", 3, 0)
text2d( fd, 155000, -76000, "O", 3, 0)
text2d( fd, 155000, -82000, "R", 3, 0)
text2d( fd, 155000, -88000, "R", 3, 0)
text2d( fd, 155000, -94000, "I", 3, 0)
text2d( fd, 155000, -100000, "D", 3, 0)
text2d( fd, 155000, -106000, "O", 3, 0)
text2d( fd, 155000, -112000, "R", 3, 0)
text2d( fd, 83500, -91000, "64N", 3, 0)
text2d( fd, 83500, -105000, "64S", 3, 0)
text2d( fd, 110000, -75500, "61", 3, 0)
text2d( fd, 113500, -106500, "62", 3, 0)
text2d( fd, 111500, -138000, "63", 3, 0)
text2d( fd, 92800, -119000, "65N", 3, 0)
text2d( fd, 87500, -135000, "65S", 3, 0)
text2d( fd, 71000, -135000, "64A", 3, 0)
text2d( fd, 31000, -70000, "R-4808", 3, 0)
text2d( fd, 29855, -85000, "W", 3, 0)
text2d( fd, 29855, -90000, "E", 3, 0)
text2d( fd, 29855, -95000, "S", 3, 0)
text2d( fd, 29855, -100000, "T", 3, 0)
text2d( fd, 30925, -127255, "S", 3, 0)
text2d( fd, 46000, -85000, "EAST", 3, 0)
text2d( fd, 22000, -61500, "PAHUTE", 3, 0)
text2d( fd, 46000, -32000, "74B", 3, 0)
```

Appendix E – EGS Database File Formats

```
text2d( fd, 51000, -57000,"74A", 3, 0)
text2d( fd, 9900, -39000,"75W", 3, 0)
text2d( fd, 22200, -48500,"75E", 3, 0)
text2d( fd, -15000, -49000,"76", 3, 0)
text2d( fd, -6000, -70000,"EC-S", 3, 0)
text2d( fd, -24097, -16200,"71N", 3, 0)
text2d( fd, -24097, -26178,"71S", 3, 0)
text2d( fd, -1324, -705,"4809", 3, 0)
text2d( fd, 8610, -20065,"EC-W", 3, 0)
text2d( fd, -7590, -5800,"A", 3, 0)
text2d( fd, 16252, -5800,"B", 3, 0)
text2d( fd, -6062, -62452,"TPEC", 3, 0)
text2d( fd, 11600, -60000,"A", 3, 0)
text2d( fd, 30619, -5800,"EC-E", 3, 0)
```

Segment # - 43

element 0

application_data(fd, 207,

```
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 204c2e20 4c2e2059 6f756e67 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472
6d617020 20206e72 616e6765 2e73666c 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })
```

remove_all_names_from_set(fd)

add_names_to_set(fd, 32,

```
{      0,      2048,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0  })
```

application_data(fd, 17,

```
{4e454c4c 49532052 45535452 49435445 44      })
```

application_data(fd, 14,

```
{41495253 50414345 20415245 4153      })
```

application_data(fd, 19,

```
{554e4954 53204152 4520494e 204d4554 455253      })
```

application_data(fd, 13,

```
{2032314d 41523834 20204c4c 59      })
```

move2d(fd, -25314.3, 4457.9)

draw2d(fd, -16514.7, 4435.5)

```
draw2d( fd, 26017, 4460.1)
draw2d( fd, 48015.6, 4559.2)
draw2d( fd, 48382.1, -16255.1)
draw2d( fd, 74600.9, -16059.4)
draw2d( fd, 74746.7, -32248.5)
draw2d( fd, 82111.8, -32178.5)
draw2d( fd, 82203.2, -41429.1)
draw2d( fd, 100636, -41225.2)
draw2d( fd, 100881, -61575.1)
draw2d( fd, 126008, -61230.9)
draw2d( fd, 127909, -155549)
draw2d( fd, 120686, -155656)
draw2d( fd, 102550, -139249)
draw2d( fd, 75699, -139535)
draw2d( fd, 71208, -137725)
draw2d( fd, 71133.8, -128481)
draw2d( fd, 26230, -128747)
draw2d( fd, 26230, -110250)
draw2d( fd, 14481.4, -110275)
draw2d( fd, -31362.1, -43628.3)
draw2d( fd, -25314.3, 4457.9)
move2d( fd, -16514.7, 4435.5)
draw2d( fd, -16537, -6666.5)
draw2d( fd, 1089.5, -32586.9)
draw2d( fd, 26133.8, -32545.6)
draw2d( fd, 26017, 4460.1)
move2d( fd, 82203.2, -41429.1)
draw2d( fd, 64507, -41586.2)
draw2d( fd, 64676.1, -63478.7)
draw2d( fd, 26230, -63718.6)
draw2d( fd, 26230, -102855)
draw2d( fd, 14481, -102855)
draw2d( fd, 14481, -110274)
move2d( fd, 100881, -61575.1)
draw2d( fd, 101125, -81923.7)
draw2d( fd, 70748.2, -82238.1)
draw2d( fd, 71133.8, -128481)
move2d( fd, 127909, -155549)
draw2d( fd, 154487, -123652)
draw2d( fd, 190212, -122913)
```

Appendix E – EGS Database File Formats

```
draw2d( fd, 241392, -38309)
draw2d( fd, 189821, -39651)
draw2d( fd, 166644, -60511)
draw2d( fd, 126008, -61231)
move2d( fd, 166644, -60511)
draw2d( fd, 167884, -123392)
move2d( fd, 153344, -60768)
draw2d( fd, 151947, 15077)
move2d( fd, 259671, -9987)
draw2d( fd, 241393, -38309)
move2d( fd, 48382.1, -16255.1)
draw2d( fd, 36582, -32581)
draw2d( fd, 26133.8, -32545.6)
move2d( fd, 36582, -32581)
draw2d( fd, 36526.7, -51007.7)
draw2d( fd, 32097.4, -51026)
draw2d( fd, 24726.1, -54751.8)
draw2d( fd, 14386.4, -54776)
draw2d( fd, 1089.5, -32586.6)
draw2d( fd, -30024, -32556)
move2d( fd, 26187, -74475)
draw2d( fd, 14386.4, -54776)
move2d( fd, 36526, -51007)
draw2d( fd, 43722, -51007)
draw2d( fd, 48015, -55068)
draw2d( fd, 48015, -63702)
move2d( fd, 137082, -61050)
draw2d( fd, 143143, -69736)
draw2d( fd, 143909, -114591)
draw2d( fd, 127819, -133345)
move2d( fd, 26230, -119488)
draw2d( fd, 42820, -128665)
line_type( fd, 3)
move2d( fd, 162942, -60582)
draw2d( fd, 161438, 16183)
move2d( fd, 5484, 4434)
draw2d( fd, 5500, -19633)
draw2d( fd, -28529, -19631)
move2d( fd, 26316, -32556)
draw2d( fd, 14560, -54863)
```

```
move2d( fd, 19615, -63665)
draw2d( fd, -17634, -63665)
move2d( fd, 38062, -63951)
draw2d( fd, 38262, -108349)
draw2d( fd, 63683, -128540)
move2d( fd, 189821, -39651)
draw2d( fd, 188464, 19536)
move2d( fd, 215607, -38980)
draw2d( fd, 167388, -98240)
move2d( fd, 48015, -55068)
draw2d( fd, 64419, -30484)
draw2d( fd, 152768, -29277)
move2d( fd, 113685, -29277)
draw2d( fd, 126008, -61231)
move2d( fd, 127688, -124104)
draw2d( fd, 90461, -124591)
draw2d( fd, 87520, -128322)
draw2d( fd, 71134, -128481)
move2d( fd, 101125, -81924)
draw2d( fd, 104964, -92974)
draw2d( fd, 105352, -124417)
draw2d( fd, 102550, -139240)
move2d( fd, 70509, -93631)
draw2d( fd, 143534, -92397)
move2d( fd, 90461, -124591)
draw2d( fd, 90304, -109795)
draw2d( fd, 105170, -109620)
move2d( fd, 83158, -139464)
draw2d( fd, 83051, -128368)
move2d( fd, 9966, -64032)
draw2d( fd, 9957, -56632)
draw2d( fd, -9249, -56632)
draw2d( fd, -9257, -64032)
move2d( fd, 9957, -56632)
draw2d( fd, 10058, -47371)
line_type( fd, 0)
move2d( fd, 48014, 4559)
draw2d( fd, -10648, 4425)
draw2d( fd, 36837, 43976)
draw2d( fd, 186479, 52806)
```


Appendix E – EGS Database File Formats

```
draw2d( fd, 188207, 30635)
draw2d( fd, 195765, 20324)
draw2d( fd, 48014, 4559)
move2d( fd, 188464, 19536)
draw2d( fd, 222013, 24078)
draw2d( fd, 270804, 8884)
draw2d( fd, 259671, -9987)
move2d( fd, 26072, -10305)
draw2d( fd, 21752, -10305)
arc( fd, 1852, 1852, 21752, -12194, 1.6, 4.7, 0, 0)
move2d( fd, 21752, -14046)
draw2d( fd, 26072, -13989)
ellipse( fd, 500, 500, 20924, -11436, 0)
```

Segment # - 44

ELEMENT 0

```
application_data( fd, 207,
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020
2f757365 72732f6d 65796572 2f747472 6d617020 2020726f 6164732e 73666c20 20202020
20202020 20202020 30202020 20202020 20302020 20202020 20203020 20202020 20202031
20202020 302020  })
remove_all_names_from_set( fd)
add_names_to_set( fd, 32,
{ 0,          4096,      0,      0,      0,      0,
0,          0,      0,      0,      0,      0,
0,          0,      0,      0,      0,      0,
0,          0,      0,      0,      0,      0,
0,          0,      0,      0,      0,      0,
0,          0 })
application_data( fd, 33,
{524f4144 532e434f 4d204649 4c452c20 55504441 54454420 31362041 50522038 32  })
application_data( fd, 26,
{524f4144 5320414e 44204e45 41524259 204c414e 444d4152 4b53  })
application_data( fd, 35,
{4c414e44 4d41524b 53204152 45204352 4f535345 532c2052 4f414453 20444153 484544
})
polyline2d( fd,
{ 4962,      -5487,      0,
5362,      -5487,      1,
```

```

        5162,      -5687,      0,
        5162,      -5287,      1}, 4, 1)
text2d( fd, 5162, -5287,"RB10", 3, 0)
polyline2d( fd,
{      1296,      -861,      0,
      1696,      -861,      1,
      1496,      -1061,     0,
      1496,      -661,      1}, 4, 1)
text2d( fd, 1496, -661,"RB4", 3, 0)
polyline2d( fd,
{      6061,      -125,      0,
      6461,      -125,      1,
      6261,      -325,      0,
      6261,       75,      1}, 4, 1)
text2d( fd, 6261, 75,"13C", 3, 0)
polyline2d( fd,
{      3611,      4027,      0,
      4011,      4027,      1,
      3811,      3827,      0,
      3811,      4227,      1}, 4, 1)
text2d( fd, 3811, 4227,"12C", 3, 0)
polyline2d( fd,
{      5976,      2056,      0,
      6376,      2056,      1,
      6176,      1856,      0,
      6176,      2256,      1}, 4, 1)
text2d( fd, 6176, 2256,"R36", 3, 0)
polyline2d( fd,
{      12465,     -14873,      0,
      12865,     -14873,      1,
      12665,     -15073,      0,
      12665,     -14673,      1}, 4, 1)
text2d( fd, 12665, -14673,"MELLAN", 3, 0) polyline2d( fd,
{      -2430,     -6259,      0,
      -2030,     -6259,      1,
      -2230,     -6459,      0,
      -2230,     -6059,      1}, 4, 1)
text2d( fd, -2230, -6059,"C.P.", 3, 0)
line_type( fd, 1)
perimeter_type( fd, 1)

```

Appendix E -- EGS Database File Formats

```
move2d( fd, 3811, 4027)
draw2d( fd, 6261, -125)
draw2d( fd, 5162, -5487)
draw2d( fd, 1496, -861)
draw2d( fd, 6261, -125)
move2d( fd, 6176, 2056)
draw2d( fd, 5084, 1959)
move2d( fd, -1836, -8073)
draw2d( fd, -2112, -5906)
draw2d( fd, 415, -1350)
draw2d( fd, 1110, -717)
draw2d( fd, 1496, -861)
draw2d( fd, 1296, -562)
draw2d( fd, 1110, -717)
move2d( fd, -1826, -8015)
draw2d( fd, 1823, -8117)
draw2d( fd, 2416, -7948)
draw2d( fd, 2934, -7943)
draw2d( fd, 5023, -5840)
draw2d( fd, 5162, -5487)
draw2d( fd, 7294, -8180)
draw2d( fd, 8316, -9010)
draw2d( fd, 11312, -8079)
draw2d( fd, 12099, -7907)
draw2d( fd, 20312, -11839)
draw2d( fd, 26072, -11839)
move2d( fd, 5162, -5487)
draw2d( fd, 9839, -3548)
draw2d( fd, 11522, -3535)
move2d( fd, -1836, -8073)
draw2d( fd, -594, -13074)
draw2d( fd, 478, -16187)
draw2d( fd, 1982.4, -20601.4)
draw2d( fd, 2662, -22482)
draw2d( fd, 3732, -27113)
draw2d( fd, 5365, -31022)
draw2d( fd, 8229, -28806)
arc( fd, 1300, 1300, 8717, -30025, 2.3, 1.1, 0, 0)
move2d( fd, 9266.4, -28846.8)
draw2d( fd, 9936, -29416)
```

```

draw2d( fd, 11765, -30117)
draw2d( fd, 14020, -30754)
draw2d( fd, 14783, -30602)
draw2d( fd, 15730, -30663)
draw2d( fd, 25420, -30084)
draw2d( fd, 24293, -19141)
draw2d( fd, 20025, -11826)
move2d( fd, -2112, -5906)
draw2d( fd, -2282, -1968)
draw2d( fd, -1935, 1934)
draw2d( fd, -1781, 3310)
draw2d( fd, -2777, 4666)
move2d( fd, -1781, 3310)
draw2d( fd, 2294, 4173)
draw2d( fd, 3811, 4027)
arc( fd, 1066, 1066, 320, -150, -0.4, 1.7, 0, 0)
move2d( fd, 134.9, 899.8)
draw2d( fd, -1781, 3310)
move2d( fd, 8316, -9010)
draw2d( fd, 7351, -10215)
draw2d( fd, 8739, -17263)
draw2d( fd, 9092, -18695)
draw2d( fd, 9350, -22806)
move2d( fd, -310, -13970)
draw2d( fd, 2139, -14200)
arc( fd, 5335, 5335, 4297, -9334, -2, -0.8, 0, 0)
move2d( fd, 7935.5, -13235.8)
move2d( fd, 6417, -229)
draw2d( fd, 8300, -1664)
move2d( fd, 6280, -140)
draw2d( fd, 8364, -1676)
line_type( fd, 0)
perimeter_type( fd, 0)

```

Segment # - 45

ELEMENT 0

application_data(fd, 207,

```

{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472

```

Appendix E – EGS Database File Formats

```
6d617020 20206c61 6b657361 2e73666c 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })

remove_all_names_from_set( fd)

add_names_to_set( fd, 32,
{
    0,      8192,    0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0,      0,      0,      0,      0,
    0,      0  })

application_data( fd, 24,
{4f55544c 494e4520 4f462041 4e54454c 4f504520 4c414b45  })

application_data( fd, 34,
{54414b45 4e204652 4f4d2031 2d363235 3030204d 41502020 3230204d 41522e20 3837  })

line_type( fd, 3)

move2d( fd, 4110, -15490)
draw2d( fd, 3920, -15550)
draw2d( fd, 3720, -15650)
draw2d( fd, 3630, -15710)
draw2d( fd, 3590, -15810)
draw2d( fd, 3595, -15920)
draw2d( fd, 3610, -16200)
draw2d( fd, 3700, -16400)
draw2d( fd, 3690, -16560)
draw2d( fd, 3780, -16800)
draw2d( fd, 3870, -16950)
draw2d( fd, 3990, -17450)
draw2d( fd, 4130, -17750)
draw2d( fd, 4230, -17840)
draw2d( fd, 4450, -18340)
draw2d( fd, 4560, -18440)
draw2d( fd, 4690, -18740)
draw2d( fd, 4840, -18890)
draw2d( fd, 4840, -19000)
draw2d( fd, 4940, -19100)
draw2d( fd, 4980, -19250)
draw2d( fd, 5300, -19450)
draw2d( fd, 5300, -19550)
draw2d( fd, 5640, -19600)
draw2d( fd, 5740, -19640)
draw2d( fd, 5940, -19500)
```

```
draw2d( fd, 6140, -19460)
draw2d( fd, 6270, -19400)
draw2d( fd, 6510, -19150)
draw2d( fd, 6670, -19120)
draw2d( fd, 6810, -18920)
draw2d( fd, 6950, -18620)
draw2d( fd, 7050, -18450)
draw2d( fd, 7100, -18200)
draw2d( fd, 6890, -17900)
draw2d( fd, 6950, -17650)
draw2d( fd, 6300, -17010)
draw2d( fd, 6000, -16600)
draw2d( fd, 5000, -16200)
draw2d( fd, 4700, -15900)
draw2d( fd, 4300, -15530)
draw2d( fd, 4110, -15490)
application_data( fd, 21,
{4f55544c 494e4520 4f462050 4544524f 204c414b 45      })
move2d( fd, 3510, -13790)
draw2d( fd, 3440, -14100)
draw2d( fd, 3480, -14500)
draw2d( fd, 3590, -14800)
draw2d( fd, 3810, -14750)
draw2d( fd, 3920, -14650)
draw2d( fd, 3950, -14550)
draw2d( fd, 4150, -14520)
draw2d( fd, 4330, -14580)
draw2d( fd, 4290, -14360)
draw2d( fd, 4420, -14350)
draw2d( fd, 4440, -14600)
draw2d( fd, 4600, -14660)
draw2d( fd, 4600, -14770)
draw2d( fd, 4750, -14570)
draw2d( fd, 4720, -14350)
draw2d( fd, 4620, -14290)
draw2d( fd, 4480, -14170)
draw2d( fd, 4370, -14070)
draw2d( fd, 4250, -13950)
draw2d( fd, 4170, -14050)
draw2d( fd, 4250, -14200)
```

Appendix E -- EGS Database File Formats

```
draw2d( fd, 4110, -14170)
draw2d( fd, 4090, -14070)
draw2d( fd, 4010, -13780)
draw2d( fd, 3780, -13700)
draw2d( fd, 3510, -13790)
application_data( fd, 30,
{4f55544c 494e4520 4c414b45 204e2e20 452e204f 4620414e 54454c4f 5045  })
move2d( fd, 4250, -15300)
draw2d( fd, 4550, -15580)
draw2d( fd, 4750, -15780)
draw2d( fd, 4900, -15840)
draw2d( fd, 5010, -15900)
draw2d( fd, 4930, -15750)
draw2d( fd, 4910, -15550)
draw2d( fd, 4810, -15350)
draw2d( fd, 4700, -15300)
draw2d( fd, 4640, -15350)
draw2d( fd, 4590, -15380)
draw2d( fd, 4490, -15410)
draw2d( fd, 4450, -15310)
draw2d( fd, 4350, -15270)
draw2d( fd, 4250, -15300)
application_data( fd, 30,
{4f55544c 494e4520 4f46204c 414b4520 4e4f5254 48204f46 20504544 524f  })
move2d( fd, 3740, -12100)
draw2d( fd, 3560, -12280)
draw2d( fd, 3600, -12580)
draw2d( fd, 3540, -12880)
draw2d( fd, 3460, -13150)
draw2d( fd, 3500, -13300)
draw2d( fd, 3520, -13680)
draw2d( fd, 3790, -13630)
draw2d( fd, 4180, -13750)
draw2d( fd, 3890, -12950)
draw2d( fd, 3920, -12690)
draw2d( fd, 3990, -12600)
draw2d( fd, 3900, -12400)
draw2d( fd, 3850, -12250)
draw2d( fd, 3740, -12100)
application_data( fd, 19,
```

```

{4f55544c 494e4520 42524f57 4e53204c 414b45  })
move2d( fd, 3780, -14850)
draw2d( fd, 3720, -14850)
draw2d( fd, 3800, -15030)
draw2d( fd, 4100, -14940)
draw2d( fd, 4100, -14940)
draw2d( fd, 4450, -14900)
draw2d( fd, 4430, -14750)
draw2d( fd, 4290, -14630)
draw2d( fd, 4000, -14650)
draw2d( fd, 3900, -14780)
draw2d( fd, 3780, -14850)
application_data( fd, 30,
{4f55544c 494e4520 534d414c 4c204c41 4b452053 4f555448 20504544 524f  })
move2d( fd, 3550, -14800)
draw2d( fd, 3510, -15050)
draw2d( fd, 3580, -15150)
draw2d( fd, 3630, -15160)
draw2d( fd, 3670, -15060)
draw2d( fd, 3630, -14920)
draw2d( fd, 3590, -14860)
draw2d( fd, 3550, -14800)
line_type( fd, 0)

```

Segment # - 46

element 0 .

```

application_data( fd, 207,
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472
6d617020 20206c61 6b65736e 2e73666c 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })
remove_all_names_from_set( fd)
add_names_to_set( fd, 32,
{
0,      16384,    0,      0,      0,      0,
0,      0,      0,      0,      0,      0,
0,      0,      0,      0,      0,      0,
0,      0,      0,      0,      0,      0,
0,      0,      0,      0,      0,      0,
0,      0 })
application_data( fd, 20,

```


Appendix E – EGS Database File Formats

```
{4f55544c 494e4520 4f46204d 41494e20 4c414b45 })
application_data( fd, 22,
{54414b45 4e204652 4f4d2031 2d363235 3030204d 4150      })
line_type( fd, 3)
move2d( fd, 0, 947)
draw2d( fd, -260, 970)
draw2d( fd, -560, 900)
draw2d( fd, -860, 770)
draw2d( fd, -1160, 630)
draw2d( fd, -1360, 560)
draw2d( fd, -1460, 430)
draw2d( fd, -1510, 300)
draw2d( fd, -1450, 100)
draw2d( fd, -1350, -30)
draw2d( fd, -1080, -130)
draw2d( fd, -950, -300)
draw2d( fd, -1040, -810)
draw2d( fd, -830, -1080)
draw2d( fd, -530, -1000)
draw2d( fd, -330, -950)
draw2d( fd, -130, -950)
draw2d( fd, 330, -990)
draw2d( fd, 530, -990)
draw2d( fd, 700, -940)
draw2d( fd, 900, -850)
draw2d( fd, 1100, -620)
draw2d( fd, 1300, -380)
draw2d( fd, 1400, 0)
draw2d( fd, 1370, 280)
draw2d( fd, 1230, 500)
draw2d( fd, 960, 700)
draw2d( fd, 540, 800)
draw2d( fd, 400, 850)
draw2d( fd, 200, 870)
draw2d( fd, 0, 947)
line_type( fd, 0)
```

Segment # - 47

ELEMENT 0

```
application_data( fd, 207,
```

```
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020 20202020 20202020
20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020 20202032 2d31302d
31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d 65796572 2f747472
6d617020 20207261 6461722e 73666c20 20202020 20202020 20202020 30202020 20202020
20302020 20202020 20203020 20202020 20202031 20202020 302020  })
```

```
remove_all_names_from_set( fd)
```

```
add_names_to_set( fd, 32,
```

```
{      0,      32768,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0,      0,      0,      0,      0,
      0,      0})
```

```
polyline2d( fd,
```

```
{      5976,      2055.80005, 0,
      6376,      2055.80005, 1,
      6176,      1855.80005, 0,
      6176,      2255.80005, 1}, 4, 1)
```

```
text2d( fd, 6176, 2255.8, "R36E", 3, 0)
```

```
polyline2d( fd,
```

```
{      -4538.8999,      -11424.9004,      0,
      -4138.8999,      -11424.9004,      1,
      -4338.8999,      -11624.9004,      0,
      -4338.8999,      -11224.9004,      1}, 4, 1)
```

```
text2d( fd, -4338.9, -11224.9, "R24", 3, 0) polyline2d( fd,
```

```
{      -2201.30005,      231.100006,      0,
      -1801.30005,      231.100006,      1,
      -2001.30005,      31.1000004,      0,
      -2001.30005,      431.100006,      1}, 4, 1)
```

```
move2d( fd, -2320, 407)
```

```
text2d( fd, -2320, 407, "R39", 3, 0)
```

```
polyline2d( fd,
```

```
{      -1700,      -2150,      0,
      -1300,      -2150,      1,
      -1500,      -2350,      0,
      -1500,      -1950,      1}, 4, 1)
```

```
move2d( fd, -2055, -1990)
```

```
text2d( fd, -2055, -1990, "R18", 3, 0)
```

```
polyline2d( fd,
```

```
{      -2910,      -9082,      0,
      -2510,      -9082,      1,
```

Appendix E – EGS Database File Formats

```
-2710,    -9282,    0,  
-2710,    -8882,    1}, 4, 1)  
text2d( fd, -2710, -8882,"R27", 3, 0)
```

Segment # - 48

ELEMENT 0

```
application_data( fd, 207,  
{46696c65 20636f6e 76657274 65642066 726f6d20 54545220 54656b74 726f6e69 7820282a  
2e636f6d 29206669 6c652e20 20202020 20202020 20202020 20202020  
20202020 20202020 20202020 20202020 20556e6b 6e6f776e 20202020 20202020 20202020  
20202032 2d31302d 31393932 20202031 20202020 30202020 20202020 2f757365 72732f6d  
65796572 2f747472 6d617020 20206d61 6e636d70 2e73666c 20202020 20202020 20202020  
30202020 20202020 20302020 20202020 20203020 20202020 20202031 20202020 302020  
})  
remove_all_names_from_set( fd)  
add_names_to_set( fd, 32,  
{    0,        65536,    0,    0,    0,    0,  
    0,        0,    0,    0,    0,    0,  
    0,        0,    0,    0,    0,    0,  
    0,        0,    0,    0,    0,    0,  
    0,        0,    0,    0,    0,    0,  
    0,        0 })  
application_data( fd, 27,  
{424f554e 44415249 45532046 4f52204d 414e4341 4d502041 524541 })  
application_data( fd, 11,  
{41554720 32342031 393832 })  
move2d( fd, -2917, 4360.2)  
draw2d( fd, -3217.7, 4409.7)  
draw2d( fd, -3168.1, 4710.3)  
draw2d( fd, -2867.5, 4660.8)  
draw2d( fd, -2875.6, 4611.6)  
draw2d( fd, -2823.6, 4603)  
draw2d( fd, -2841, 4497.1)  
draw2d( fd, -2893, 4505.7)  
draw2d( fd, -2917, 4360.2)
```

III. Starbase 'Bitmap' File Format

A. Format on Disk (Frame Buffer)

A Starbase 'bitmap' is defined as "a rectangular array of pixels" which represent the memory image (or portion) of the current display. In essence, the bitmap is the current frame buffer image.

On the EGS system the bitmap is stored in 'Pixel-Major' format, which means the entire 8-bit byte (8 image planes) represents a single screen pixel.

Only those image planes that are display-enabled are saved to the file. The bits corresponding to planes which are not currently displayed are set to 0's in the file.

The current color map is stored with any bitmap file (in EGS) .

A file header is included with each bitmap file and includes the following information. Refer to 'bitmapfile(4)' in the "Starbase Reference" manual for a detailed description of the file format.

```
struct bf_header {
char fileID[16] ;           /* "BitmapFile" name */
    int Revision ;          /* revision of raster file format */
    char DeviceType[16] ;    /* device that stored this file */
    int Xstart, Ystart ;     /* upper left corner of original */
    int Xlen, Ylen ;         /* size of pixel block in file */
    int BitmapMode ;         /* -1 = pixel-major mode */
    int Depth ;              /* 8 planes */
    int PixelAlignment ;     /* 8 - every byte = pixel */
    int NumberOfBanks ;      /* 1 bank in file */
    int ColorMapMode ;       /* 0 - NORMAL */
    int ColorMapSize ;       /* number of color map entries */
    int BackgroundIndex ;    /* color map index of background */
}
```

B. Saving the 'Bitmap' File

Use the Starbase 'bitmap_to_file' command (Page 15-9).

Example Command Format:

```
/* --- store the screen image on file ----- */
bitmap_to_file(fildes,          /* device whose image is to be stored */
               TRUE,            /* store ALL planes... */
               0,               /* ignore - source plane */
               0,               /* destination plane */
               "./BitmapFile",  /* file name in current directory */
               TRUE,            /* store the whole screen... */
               0.0, 0.0,        /* ignore - upper left corner */
               0, 0,            /* ignore - size to save */
               TRUE ) ;         /* store color map */
```

C. Reading The 'Bitmap' File

1. Reading the File Header

Use the Starbase 'inquire_file' command (Page 15-14).

Example Command Format:

```
/* --- interrogate the 'bitmap' file ----- */
struct bf_header {
    char fileID[16] ;           /* "BitmapFile" name */
    int Revision ;             /* revision of raster file format */
    char DeviceType[20] ;      /* device that stored this file */
    int Xstart, Ystart ;       /* upper left corner of original */
    int Xlen, Ylen ;           /* size of pixel block in file */
    int BitmapMode ;           /* -1 = pixel-major mode */
    int Depth ;                /* 8 planes */
    int PixelAlignment ;       /* 8 - every byte = pixel */
    int NumberOfBanks ;        /* 1 bank in file */
    int ColorMapMode ;         /* 0 - NORMAL */
    int ColorMapSize ;         /* number of color map entries */
    int BackgroundIndex ;      /* color map index of background */
} bf ;

/* --- bitmap file name = "BitmapFile" (Current Directory) ----- */
inquire_file("./BitmapFile", &bf.Revision, bf.DeviceType, &bf.Xstart,
    &bf.Ystart, &bf.Xlen, &bf.Ylen, &bf.BitmapMode, &bf.Depth,
    &bf.PixelAlignment, &bf.NumberOfBanks, &bf.ColorMapMode,
    &bf.ColorMapSize, &bf.BackgroundIndex) ;
```

2. Reading the 'Bitmap' File

Use the Starbase 'file_to_bitmap' command (Page 15-11).

Example Command Format:

```
/* --- retrieve the image from the file ----- */
file_to_bitmap(fildes,        /* device whose image is to be updated */
    TRUE,                    /* retrieve ALL planes in file... */
    0,                       /* ignore - source plane */
    0,                       /* destination plane */
    "./BitmapFile",          /* file name in current directory */
    0.0, 1.0,                /* upper left corner in VDCs */
    TRUE ) ;                 /* get store color map from file */
```

D. Printing the Screen Image

Three possible Starbase methods to obtain graphics hardcopy are the 'bitmap_print', 'file_print', and off-line (HP-UX) print modes. Each of these methods is further described in this section.

In each case the Printer Command Language (PCL) formatter is used to convert the bitmap image to the printer format.

Configuration files for each method are included at the end of this section. These configuration files are described in the PCL formatter section of the "Starbase Device Drivers Library Manual, Volume 2".

When printing a bitmap image, printer 'spooling' is recommended to avoid locking the application program with the graphics output operation.

1. Method 1

Spooling from a Starbase program using the 'bitmap_print' function call. It uses a printer disk file (myprint.proc) as a buffer to avoid slowing the program down with printer I/O.

Configuration File name: "config.prsp"

Example Commands:

```
/* format the file - output filename is 'myprint.proc' */
bitmap_print(fildes,          /* the fildes of the bitmap to dump */
             "pcl",          /* use the PCL formatter */
             "./config.prsp", /* configuration file to use */
             ALL_PLANES,     /* print all planes in bitmap */
             TRUE,           /* whole bitmap is to be printed */
             0.0, 0.0,       /* ignore - upper left corner */
             0, 0,           /* ignore - size of bitmap to print */
             FALSE,          /* don't rotate the image */
             0, 0,           /* ignore - fore/background colors */
             TRUE ) ;        /* don't print a background color */
system("lp -oraw myprint.proc") ; /* spool file to printer */
```

2. Method 2

Spooling from a Starbase program using the 'file_print' function call. It allows the application to select a disk file (myprint.dat) in which to save the bitmap. It still uses the printer disk file (myprint.proc) as a buffer to avoid slowing the program down with printer I/O.

Configuration File name: "config.prsp"

Example Commands:

```

/* create the Starbase bitmap file */
bitmap_to_file(fildes, TRUE, 0, 0, "./BitmapFile", 0.0, 0.0, 0, 0, TRUE) ;
/* format the file, output filename is myprint.proc*/
file_print("./BitmapFile", /* name of the file to print */
           "pcl",           /* use the PCL formatter */
           "./config.prsp", /* configuration file to use */
           ALL_PLANES,      /* print all planes in bitmap */
           FALSE,           /* don't rotate the image */
           0, 0,            /* ignore - fore/background colors */
           TRUE ) ;        /* don't print a background color */
/* spool the file */
system("lp -oraw myprint.proc") ; /* spool file to printer */

```

3. Method 3

Spooling using the HP-UX command "pcltrans". This method assumes that a Starbase application program has previously created a bitmap file 'myprint.dat' and it resides in the current directory.

Configuration File name: None

Example Command:

```
pcltrans myprint.dat | lp -oraw
```

E. PCL formatter Configuration File (Spooled)

```

# /* STARBASE_ID:config.prsp (spooled)      10/16/91   11:10:00 */
# -----
#       This is a configuration file for the PCL formatter
#       a # begins a comment
# -----
# -- Output goes to std out -- uncomment one item
#
FALSE   #   Output is not to std out
# TRUE   #   Output is to std out
# -----
# -- Special device file or output file name
#       edit as necessary - this field is ignored
#       when output to std out is enabled but it

```

```

#          MUST be present
#
./myprint.proc # This is the special device or output file
# -----
# -- Print Method -- uncomment one item
#
# gray      # Use gray scale dithering
# mono      # All non-zero pixels will be printed black
color        # use color mode (RGB & CYM)
# -----
# -- Pixel expansion -- edit as necessary (range is 1..8)
#
1            # This is the pixel expansion to be used
# -----
# -- Resolution in dots per inch -- uncomment one item
#
# 90         # Resolution is 90 dpi 293X family printers
# 96         # Resolution is 96 dpi 222X family printers
# 140        # Resolution is 140 dpi 256X family printers
180         # Resolution is 180 dpi 3630 family printers
# 192        # Resolution is 192 dpi 222X family printers
# 300        # Resolution is 300 dpi 2686 family printers
# -----
# -- Send Resolution - uncomment one item
#
# FALSE      # Do not send graphics resolution to printer
TRUE         # Send graphics resolution to printer
# -----
# -- Print start position -- uncomment one item
#
margin       # Normal mode start graphics at left margin
# current    # Start graphics at current position (this
#             also suppresses form-feed at end of print)
#
# -----
# -- Page Width in inches -- edit as necessary
#
8.0          # This is the page width in inches (printable area)
# 10.5       # This is the page width in inches (printable area) # -----
# -----
# -- Page Length in inches -- edit as necessary

```



```
#
10.5 # This is the page length in inches (printable area)
# 13.5 # This is the page length in inches (printable area)
# -----
```

F. PCL formatter Configuration File (Direct Access)

```
# /* STARBASE_ID:config.prtr          10/16/91   08:15:00 */
# -----
#       This is a configuration file for the PCL formatter
#       a # begins a comment
# -----
# -- Output goes to std out -- uncomment one item
#
FALSE          #       Output is not to std out
# TRUE        #       Output is to std out
# -----
# -- Special device file or output file name
#       edit as necessary - this field is ignored
#       when output to std out is enabled but it
#       MUST be present
#
/dev/rlp3630    # This is the special device or output file # -----
# -----
# -- Print Method -- uncomment one item
#
# gray        # Use gray scale dithering
# mono        # All non-zero pixels will be printed black
color          # use color mode (RGB & CYM)
# -----
# -- Pixel expansion -- edit as necessary (range is 1..8)
#
1              # This is the pixel expansion to be used
# -----
# -- Resolution in dots per inch -- uncomment one item
#
# 90          # Resolution is 90 dpi   293X family printers
# 96          # Resolution is 96 dpi   222X family printers
# 140         # Resolution is 140 dpi  256X family printers
180          # Resolution is 180 dpi  3630 family printers
# 192         # Resolution is 192 dpi  222X family printers
# 300         # Resolution is 300 dpi  2686 family printers
```

```

# -----
# -- Send Resolution - uncomment one item
#
# FALSE # Do not send graphics resolution to printer
TRUE      # Send graphics resolution to printer
# -----
# -- Print start position -- uncomment one item
#
margin      # Normal mode start graphics at left margin
# current # Start graphics at current position (this
#   also suppresses form-feed at end of print)
#
# -----
# -- Page Width in inches -- edit as necessary
#
8.0    # This is the page width in inches (printable area)
# 10.5  # This is the page width in inches (printable area) # -----
# -----
# -- Page Length in inches -- edit as necessary
#
10.5    # This is the page length in inches (printable area)
# 13.5  # This is the page length in inches (printable area)
# -----

```


APPENDIX F

EGS/TTR Coordinate System (NAD27) Equations

References:

1. "Spherical and Practical Astronomy as Applied to Geodesy" by Ivan I. Mueller; Frederick Ungar Publishing Company, 1969, New York.
2. "Satellite Surveying - Theory Geodesy Map Projections" by Gregory J. Hoar; Magnavox Applications Equipment Operations, 1982, Magnovox - Marine Systems Division.
3. "Optimal Coordinate Transformation Formulas" by D. K. Olson; Technical Note# 3450-1-77; 1977, Data Processing Division / Range Instrumentation Systems Department, Pacific Missile Center, Point Mugu, California.
4. "Global Coordinate System"; Document# 103-69, Data Reduction & Computing Working Group; published by Secretariat Range Commanders Council White Sands Missile Range, New Mexico 88002.
5. "Coordinate Transformations Used in Data Reduction at Tonopah Test Range" by Lestor W. Lathrop; SAND78-2233, 1978, Sandia National Laboratories, Albuquerque, New Mexico, 87185.
6. "Department of Defense World Geodetic System 1984"; DMA Stock No. DMA TR83502WGS84, DMA Technical Report DMA TR 8350.2, Second Edition, 1 September 1991; The Defense Mapping Agency, Systems Center(SG), 8613 Lee Highway, Fairfax, Virginia 22031-2138.

Parameters (NAD27) : Ellipsoid

Semi-major axis - $A = 6,378,206.4$ meters (20,925,874.02 feet)
 Semi-minor axis - $B = 6,356,583.8$ meters (20,854,933.73 feet)
 Eccentricity(squared) - $e^2 = (1 - b^2/a^2) = .00677$
 coef. of flattening - $f = (1 - b/a) = .00339 = 1 / 294.9787$

Parameters (WGS84) : Ellipsoid

Semi-major axis - $A = 6,378,137.0$ meters (20,925,646.33 feet)
 Semi-minor axis - $B = 6,356,752.3$ meters (20,855,486.55 feet)
 Eccentricity(squared) - $e^2 = (1 - b^2/a^2) = .00669$
 coef. of flattening - $f = (1 - b/a) = .00335 = 1 / 298.2572$

TTR range zero coordinates (NAD27) : (WGS84) :

Latitude	= 37°50'36.71085"	Latitude	= 37 50'36.503"
Longitude	= -116°43'44.37431"	Longitude	= -116 43'47.600"
height(msl)	= 0.0 feet	height(msl)	= 0.0 feet

I. Range system (x,y,z)

Range zero: $x_0 = 0.0$; $y_0 = 0.0$; $z_0 = 0.0$

Below main lake target - 5331.34 feet (1624.9924 meters)

Slant Range: $R = \sqrt{x^2 + y^2 + z^2}$

Azimuth : $A = \arctan(x/y)$; degrees from north - clockwise (+)
(0 -> 360 degrees)

Elevation : $E = \arctan(z / \sqrt{x^2 + y^2})$;
degrees above x-y plane (+)
(+ 90 degrees)

and

$$\begin{aligned} x &= R \cos E \sin A ; && \text{east component} \\ y &= R \cos E \cos A ; && \text{north component} \\ z &= R \sin E ; && \text{vertical component} \end{aligned}$$

II. Master Coordinate System (E,F,G)

Converting to/from the master (earth-centered) coordinate from/to range coordinates involves a rotation and a translation.

A. In NAD27 the translations are:

$$E0 = -2268297.9774 \text{ meters (calculated values)}$$

$$F0 = -4504328.8599 \text{ meters}$$

$$G0 = 3891549.6278 \text{ meters}$$

$$\begin{bmatrix} E0 \\ F0 \\ G0 \end{bmatrix} = \begin{bmatrix} (N_0 + h_0) \cos \phi_0 \cos \lambda_0 \\ (N_0 + h_0) \cos \phi_0 \sin \lambda_0 \\ [N_0 (1-e^2)] \sin \phi_0 \end{bmatrix}$$

where

$$N_0 = \frac{A}{\sqrt{1 - e^2 \sin^2 \phi_0}} = 6386346.716 \text{ meters}$$

B. The rotation matrices are :

$$[R \lambda_0] = \begin{bmatrix} -\sin \lambda_0 & -\cos \lambda_0 & 0 \\ \cos \lambda_0 & -\sin \lambda_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} ; \quad \begin{array}{l} \text{(Rotate range coord.} \\ \text{parallel to longitude} \\ \text{axis (Greenwich))} \end{array}$$

$$[R \phi_0] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sin \phi_0 & -\cos \phi_0 \\ 0 & \cos \phi_0 & \sin \phi_0 \end{bmatrix}; \quad \begin{array}{l} \text{(Rotate range coord.} \\ \text{parallel to equatorial plane)} \end{array}$$

therefore the complete transformation is:

$$\begin{bmatrix} E \\ F \\ G \end{bmatrix} = [R\lambda_0][R\phi_0] \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} E0 \\ F0 \\ G0 \end{bmatrix};$$

and the inverse transformation is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [R\phi_0]^T [R\lambda_0]^T \begin{bmatrix} E - E0 \\ F - F0 \\ G - G0 \end{bmatrix};$$

III. Geodetic Coordinate System (ϕ, λ, h)

Note: $h = H(\text{MSL}) + \varepsilon$;

where $\varepsilon = -18.2$ meters at TTR

$$N = \frac{A}{\text{sqrt}(1 - e^2 \sin^2 \phi_0)}$$

$$\begin{bmatrix} E \\ F \\ G \end{bmatrix} = \begin{bmatrix} (N+h) \cos \phi \cos \lambda \\ (N+h) \cos \phi \sin \lambda \\ [N(1-e^2)] \sin \phi \end{bmatrix}$$

Latitude : ϕ = is found by iteration (see references # 4 & 5) ;

Longitude : $\lambda = \arctan(F/E)$;

Height : $h = \frac{\text{sqrt}(E^2 + F^2)}{\cos \phi} - N$;

IV. Standard Molodensky Datum Transformation Formulas* (reference #6)

$$d\phi = \{-dX \sin \phi \cos \lambda - dY \sin \phi \sin \lambda + dZ \cos \phi \\ + da (R_N e^2 \sin \phi \cos \phi) / a + df [R_M (a/b) + R_N (b/a)] \sin \phi \cos \phi\} \\ [(R_M + h) \sin 1'']^{-1}$$

$$d\lambda = \frac{-dX \sin \lambda + dY \cos \lambda}{(R_N + h) \cos \phi \sin 1''}$$

$$dh = dX \cos \phi \cos \lambda + dY \cos \phi \sin \lambda + dZ \sin \phi - da (a/R_N) \\ + df (b/a) R_N \sin^2 \phi$$

*WARNING: Don't use above equations between 89 Latitude and the pole.

where:

ϕ = old geodetic latitude (radians).

λ = old geodetic longitude (radians).

h = old ellipsoid height (meters).

$d\phi, d\lambda, dh$ = corrections to transform "from" geodetic coordinates "to" new geodetic datum coordinates.

dX, dY, dZ = shifts between centers of "from" datum and "to" datum.

for NAD27 to WGS84 datum translation:

$$dX = -7.9 \text{ meters}$$

$$dY = 159.7 \text{ meters}$$

$$dZ = 175.5 \text{ meters}$$

a = semi-major axis of "from" Datum.

b = semi-minor axis of "from" Datum.

f = flattening of "from" Datum.

da, df = differences between the semi-major axis and flattening of "to" Datum minus "from" Datum.

for NAD27 to WGS84 datum translation:

$$da = 6378137.0 - 6378206.4 = -69.4 \text{ meters}$$

$$df = .00335 - .00339 = -0.00004$$

e^2 = first Eccentricity Squared of "from" Datum.

$$e^2 = 2f - f^2$$

IV. Standard Molodensky Datum Transformation Formulas*

R_N = radius of curvature in the prime vertical.

$$R_N = \frac{a}{(1 - e^2 \sin^2 \phi)^{1/2}}$$

R_M = radius of curvature in the meridian.

$$R_M = \frac{a(1 - e^2)}{(1 - e^2 \sin^2 \phi)^{3/2}}$$

The New "to" Datum coordinates are:

$$\phi' = \phi + d\phi$$

$$\lambda' = \lambda + d\lambda$$

$$h' = h$$

APPENDIX G

**Building Segment Files (*.sfl)
from Tektronix Type Files (*.com)**

Note: The following discussion assumes the user is in the integrated 'X-window' environment.

1. Use the 'vi' editor with Graphics Refresh Command List (XREF/Tektronix) commands to create/edit the source file.
 - a. Enter the hp terminal window 'login' and enter your (egs) password. (or create a 'New Window' from the Root Window)
 - b. The output file should be a '*.com' file in the directory '/users/egs/data'.
 - c. Terminate each command line with a '\$' symbol.
 - d. Terminate the file with a '%' symbol.
 - e. Use the Graphics Refresh Editor Commands to construct the new segment.

example: `cd /users/egs/data`

`vi filename.com`

`<Graphics Refresh Editor Commands>$`

`%`

`:wq`

2. Use the 'ShowSeg' program to convert the Tektronix format (*.com) to the Egs Starbase (*.asc) format (and/or the *.sfl format).
 - a. Use the Mouse to Restore (double click) the 'ShowSeg' icon. (See Figure 2.0)
 - b. Fill in the entry 'Segment Filename(*.asc)' with the same "filename.com" name as used in step 1.
 - c. Complete the remaining entries of the form as desired.
 - d. Using the Mouse, select (click) on the "SHOW_SEG" button.

Note: If a "filename" already exists you will be asked if it is OK to overwrite it (Answer Yes or No).

- e. Use the Mouse to select (click) on the "QUIT" button when finished. Answer "NO" when asked to save the default file.

Building Segment Files (*.com)
GRAPHICS REFRESH EDITOR COMMAND LIST

Note: '\$' symbol terminates any command.
 '%' symbol ends the "*.com" file.

<u>COMMAND</u>	<u>ARGUMENTS</u>	<u>FUNCTION</u>	<u>EXAMPLE</u>
CIRCLE	XC YC RADIUS ANG1 ANG2	Draw a circle or arc centered at XC YC with radius=RADIUS beginning at ANG1 and ending at ANG2. An arc is drawn from ANG1 to ANG2 in the direction of the sign of (ANG2 - ANG1) (plus -> counterclockwise / negative -> clockwise)	CIRCLE 0 0 10 0 360\$
CLINE	X Y HEADING	Courseline: Draw a line through Point(X,Y) at the angle defined by HEADING of length defined by the window limits	CLINE 2136 306 170\$
CROSS	X Y	Draw a cross at window coordinates (X,Y)	CROSS 254.3 -1097.6\$
DASH	None	Use DASH line-type for all subsequent lines	DASH\$
DEFWIN	None	Set window to default value (-16510, -21590,21590,16510) meters. This is the range m dimensions in meters.	DEFWIN\$
VTSCA	Constant	Change test character size. Default size is 3	VTSCA 3\$
WDRAW	X Y	Draw from current window position to window position (X,Y).	WDRAW -4123 5678\$
WGRID	XMIN YMIN XMAX YMAX DX DY	Draw a grid starting at XMIN, YMIN and ending at XMAX, YMAX with spacing DX and DY. DX and DY do not have to be the same.	WGRID 0 0 10000 10000 1000 1000\$
WIND	XMIN YMIN XMAX YMAX	Set window to a rectangle beginning at XMIN, YMIN with diagonal XMAX, YMAX	WIND - 5000 -5000 5000 5000\$
WINDRA	None	Set the window to range map default dimensions: XMIN = -16510 meters YMIN = -21590 meters XMAX = 21590 meters YMAX = 16510 meters	WINDRA\$

Building Segment Files (*.com)
GRAPHICS REFRESH EDITOR COMMAND LIST

<u>COMMAND</u>	<u>ARGUMENTS</u>	<u>FUNCTION</u>	<u>EXAMPLE</u>
WINDSE	None	Set the window to section map default dimensions: XMIN = -259000 meters YMIN = -315000 meters XMAX = 301000 meters YMAX = 245000 meters	WINDSES
WMOVE	X Y	Move from current window space position to window space position (X,Y).	WMOVE 1000 2000\$
WRECT	XMIN YMIN XMAX YMAX	Draw a window space rectangle from XMIN, YMIN to diagonal position XMAX, YMAX.	WRECT 0 0 1500 2000\$
REMARK	Character String	Allows comments (remarks) to be put in a command file for file annotation	REMARK 'created 30JAN92'\$
UNITS	'FEET' or 'METRIC'	Sets units I/O display mode for all subsequent input	UNITS METRIC\$
VDRAW	IX IY	Draw from the current view-space device coordinates to device coordinates IX, IY.	VDRAW 200 500\$
VMOVE	IX IY	Move from the current view-space device coordinates to position IX, IY.	VMOVE 5000 5000\$
VRECT	IXMIN IYMIN IXMAX IYMAX	Draw a view space rectangle from IXMIN, IYMIN to diagonal IXMAX, IYMAX (device coordinates).	VRECT 0 0 5000 5000\$
VTEXT	Character String	Draw text string beginning at the current position. String must be enclosed by single quotes.	VTEXT 'ANT TGT'\$
LINE	None	Draw subsequent vectors in SOLID line mode.	LINES
PATT	3 Hex Characters	Change line type mode 05F => LONG_DASH 03F => DASHED_DOT 02F => DOT 00F => SOLID	PATT 03F\$

Figure G.1. Tektronix 4081 Viewports

Figure G.2. Building a Segment file (*.sfl) using the Tektronix text files (*.com)

Segment File Names: *.asc *.sfl

Remarks:

Switch Number: Segment Name:

Author:

Window Units:

Window Coordinates (lower, left) =

(upper, right) =

Figure G.3. 'Show_Seg' Setup Window

View Filenames: (X asc) (X vfl)

Remarks:

Switch Number: View Name:

Author:

Window Units:

Window Coordinates (lower, left) X: Y:

(upper, right) X: Y:

Segment #1 name:

Segment #2 name:

Segment #3 name:

Segment #4 name:

Segment #5 name:

Segment #6 name:

Segment #7 name:

Segment #8 name:

Figure G.4. 'Build' Setup Window

APPENDIX H

Building Segment Files (*.sfl) from Starbase/Display List Type Files (*.asc)

Note: The following discussion assumes the user is in the integrated 'X-window' environment.

1. Use the 'vi' editor with EGS Starbase/DL Command List commands to create/edit the source file.
 - a. Enter the hp terminal window 'login' and enter your (egs) password. (or create a 'New Window' from the Root Window)
 - b. The output file should be a '*.asc' file in the directory '/users/egs/data'.
 - c. Terminate each command line with a ';' symbol.
 - d. Use the ':r' read file command to input the initial header template information from file 'header.asc'. Important, this header must be the first entry in the file.
 - e. Edit this header template for the current file data. Note, when editing the 'application_data' lines use only the 'REPLACE MODE'.
 - f. Use the EGS Starbase/DL Segment Commands to construct the new segment.

example: `cd /users/egs/data`

```
vi filename.asc
r header.asc
<Edit the header information>
.
.
<EGS Starbase/DL Segment Commands>;
.
.
:wq
```

2. Use the 'ShowSeg' program to view the file and check for proper syntax. (and/or convert the ASCII text format (*.asc) to the Starbase segment (*.sfl) format).
 - a. Use the Mouse to Restore (double click) the 'ShowSeg' icon. (See Figure H.2)
 - b. Fill in the entry 'Segment Filenames (*.asc)' with the same "filename.asc" name as used in step 1.
 - c. Edit the remaining entries of the form as desired (or use the Mouse to select (click) the "Seg Info." button which loads header information).
 - d. Using the Mouse, select (click) on the "SHOW_SEG" button.

Note: If a "filename" already exists you will be asked if it is OK to overwrite it (Answer Yes or No).

- e. Use the Mouse to select (click) on the "QUIT" button when finished. Answer "NO" when asked to save the default file.
3. Use the 'Build' program to build views for the Real-Time graphics program.

4

•

- 9

-

Figure H.1. Building a View file (*.vfl) using the Starbase text files (*.asc)

Appendix H – Building Segment Files (*.sfl) from Starbase/Display List Type Files (*.asc)

```

application_data(3, 207,data)
123456789012345678901234567890123456789012345678901234567890123456789
AuthorAAAAAAAAAAAAAA' DateDDDDDD Rev# Swh# Vnam PathPPPPPPPPPPPPPPPPPP
FilenameFF
FFF      0      0      0      1      0  ;
remove_all_names_from_set( 3);
add_names_to_set( 3, 32, NameSet)
    2,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,
    0,    0;
dl_label( 3, 1);
view_window( 3,-16510.0,-21590.0, 21590.0, 16510.0);

```

Caution: The 'application_data' block must be exactly 207 bytes long (excluding terminating 'CR' & ';').
The location of each byte in the array is critical.

1. The 'application_data' block may be edited with the 'vi' editor but the 'REPLACE' mode must be used. Don't use the 'INPUT' or 'DELETE' modes.

Line 2: Comment Line (80 characters Maximum)

Replace characters '123 ... 890' with any 'Comments or Remarks' Use spaces ' ' to overwrite (replace) unused characters.

Line 3: Author	(20 characters Maximum) - file creator
Date	(10 characters Maximum) - last date revised
Revision Number (Rev#)	(4 characters Maximum) - should be a number
Switch Number (Swh#)	(4 characters Maximum) - should be a number [1 -> 8]
View Name (Vnam)	(4 characters Maximum) - View Name (Switch Label) which is passed to the Control Panel.
Path	(21 characters Maximum) - directory pathname where the 'segment' file resides.
Filename	(15 characters Maximum) - filename of this 'segment' (*.sfl).

2. Leave the 'remove_all_names_from_set' and 'add_names_to_set' commands alone they are just placeholders which are updated later by the real-time program.
3. Edit the values in the 'dl_label' and 'view_window' commands as desired, but the commands must remain in the header.

Segment File Name:

Remarks:

Switch Number: Segment Name:

Author:

Window Units:

Window Coordinates (lower left):

(upper right):

Figure H.2. "Show_Seg" Setup Window

View Filenames (*asc): (*vfl):

Remarks:

Switch Number: View Name:

Author:

Window Units:

Window Coordinates (lower, left) X: Y:

(upper, right) X: Y:

Segment #1 name:

Segment #2 name:

Segment #3 name:

Segment #4 name:

Segment #5 name:

Segment #6 name:

Segment #7 name:

Segment #8 name:

Figure H.3. "Build" Setup Window

EGS Starbase/display List Segment (*.asc) Commands

1. ';' symbol terminates any command.
2. Refer to Figure H.4

All 'world coordinates' are in meters and are user defined by the last 'view_window' command.

All 'virtual coordinates' are by default

range	zero location
x: 0.0 -> 1.25 y: 0.0 -> 1.00	<0.0, 0.0> is lower, left corner

The 'virtual coordinate' window system is fixed at Setup Time in the WINDEF file.

All 'device coordinates' are screen points on the display device. For the HP98550 graphics display:

range	zero location
x: 0 -> 1279 points y: 0 -> 1023 points	< 0. 0 > is upper, left corner

3. All angle units must be in radians.
4. The file descriptor variable "fd" in the following discussion is just a placeholder. The actual value depends on the window in which the segment is displayed and is set at runtime. By default "fd" equals 3 in these examples, but any integer number may be used.
5. Figure H.5 shows the HP98550A default color indices and equivalent RGB values.
6. Figure H.6 shows the default marker types available.
7. Figure H.7 shows the default line (perimeter) types available.

Figure H.4. EGS Starbase Viewports

COMMAND

FUNCTION

add_names_to_set

Add names to the current name set. This is used by the 'invisibility' filter when turning segments on or off. It is a 'placeholder' in the file segment format and must exist. It should not be changed by the user.

application_data

Save data (bytes) in the currently open display segment. This data is ignored during graphics display but may be saved/recovered by a users program.

format: application_data(fd, count, data)
 int fd, count;
 char "data";
 char array[count];

example: application_data(3, 30, data)
 Test Race Track - July 9, 1992;

arc

Draw an elliptical or circular arc centered at <xc,yc> with radii <xrad,yrad> beginning at <ang1> and ending at <ang2>. <rot> specifies the amount the arc is to be rotated in radians. <close> specifies how the arc is closed per:

0 -> No Chord

1 -> Pie closure

2 -> Chord

also ang2 > ang1 -> draw counterclockwise

 ang2 < ang1 -> draw clockwise

format: arc(fd, xrad, yrad, xc, yc, ang1, ang2, rot, close);
 int fd;
 float xrad, yrad, xc, yc, ang1, ang2, rot;
 int close;

example: arc(3,1000.0,1000.0,500.0,1000.0,0.0,3.1416,0.0,0);

COMMAND

FUNCTION

background_color

Set background color by color table <index> or RGB intensity values <red,green,blue>.

formats: background_color_index(fd, index);
 int fd, index;
 background_color(fd, red, green, blue);
 int fd;
 float red, green, blue;

examples (white): background_color_index(3, 1);
 background_color(3, 1.0, 1.0, 1.0);

character_height

Set character height in 'virtual' <height> or 'device' <dcheight> coordinates.

formats: character_height(fd, height);
 int fd;
 float height;

 dccharacter_height(fd, dcheight);
 int fd, dcheight;

examples: character_height(3, .02);
 dccharacter_height(3, 20);

dl_label

Insert a display list label in the currently open display segment. It is just a 'placeholder' in the segment file format and identifies a location in the file. The label must exist. It is used in editing the file in real-time and in building the 'view' files.

format: dl_label(fd, seg_id);
 int fd, seg_id;

example: dl_label(3, 1);

draw

Draw a line from the current to the specified pen position in 'world' <wcx,wcy> or 'device' coordinates <dcx,dcy>.

formats: draw2d(fd, wcx, wcy);
 int fd;
 float wcx, wcy;

 dcdraw(fd, dcx, dcy);
 int fd, dcx, dcy;

examples: draw2d(3, 1000.0, 5000.0);
 dcdraw(3, 535, 393);

COMMAND

FUNCTION

ellipse

Draw an ellipse or circle centered at <xc,yc> with radii <xrad,yrad>. <rot> specifies the amount the ellipse is to be rotated in radians.

format: ellipse(fd, xrad, yrad, xc, yc, rot);
int fd;
float xrad, xc, yc, rot;

example: ellipse(3, 1000.0, 1000.0, 500.0, 1000.0, 0.0);

fill_color

Set color table <index> or RGB intensity values <red, green,blue> for filled areas (ellipse, rectangle, polygon, etc.).

formats: fill_color_index(fd, index);
int fd, index;

fill_color(fd, red, green, blue);
int fd;
float red, green, blue;

examples (red): fill_color_index(3, 2);

fill_color(3, 1.0, 0.0, 0.0);

hatch_orientation

Specify hatch line orientation where: hatch angle = arctan(vx/vy)

format: hatch_orientation(fd, vx, vy);
int fd;
float vx, vy;

example (45 degrees): hatch_orientation(3, 1.0, 1.0);

hatch_spacing

Specify spacing <space> between hatch lines in <mode> units follows:

0 -> VDC_UNITS (virtual coordinates - 0.0 -> 1.25)
1 -> WC_UNITS (world coordinates - meters)
2 -> DC_UNITS (device coordinates - points)

format: hatch_spacing(fd, space, mode);
int fd;
float space;
int mode;

example: hatch_spacing(3, .031, 0);

<u>COMMAND</u>	<u>FUNCTION</u>
hatch_type	<p>Specify type <style> of hatching to be done as follows: 0 -> PARALLEL_HATCH 1 -> CROSSHATCH</p> <p><u>format</u>: hatch_type(fd, style); int fd, style;</p> <p><u>example</u>: hatch_type(3, 0);</p>
interior_style	<p>Select fill type <style> and boundary visibility <edged> for subsequent filled area primitives and polygons.</p> <p><style> - 0 -> INT_HOLLOW (not filled) 1 -> INT_SOLID (filled) 2 -> INT_HATCH (hatched)</p> <p><edged> - 0 -> FALSE (not edged) 1 -> TRUE (edged)</p> <p><u>format</u>: interior_style(fd, style, edged); int fd, style, edged;</p> <p><u>example</u>: interior_style(3, 0, 1);</p>
line_color	<p>Select color index <index> or RGB color intensity value <red, green, blue> for subsequent line primitives.</p> <p><u>formats</u>: line_color_index(fd, index); int fd, index;</p> <p>line_color(fd, red, green, blue); int fd; float red, green, blue;</p> <p><u>examples</u> (green): line_color_index(3, 4); line_color(3, 0.0, 1.0, 0.0);</p>

COMMAND

FUNCTION

line_type

Select line type <style> for all subsequent line primitives.
(See Line Types)

format: line_type(fd, style);
int fd, style;

example (solid): line_type(3, 0);

line_width

Set line width <width> in the following coordinates:
<mode> - 0 -> VDC_UNITS (virtual coordinates - 0.0 -> 1.0)
1 -> MC_UNITS (world coordinates - meters)

format: line_width(fd, width, mode);
int fd, mode;
float width;

example: line_width(3, .040, 0);

marker_color

Set color table <index> or RGB color intensity value <red, green, blue> for subsequent polymarker primitives.

formats: marker_color_index(fd, index);
int fd, index;

marker_color(fd, red, green, blue);
int fd;
float red, green, blue;

examples (blue): marker_color_index(3, 6);

marker_color(3, 0.0, 0.0, 1.0);

marker_orientation

Define orientation of symbols drawn with marker primitives
where: orientation angle = $\arctan(vx / vy)$

format: marker_orientation(fd, vx, vy);
int fd;
float vx, vy;

example (0 degrees): marker_orientation(3, 0.0, 1.0);

COMMAND

FUNCTION

marker_size

Set polymarker size <size> per <mode> as follows:

<mode> - 0 -> virtual coordinates (0.0 -> 1.0)

1 -> device dependent scale factor (?)

formats: marker_size(fd, size, mode);
int fd, mode;
float size;

dcmarker_size(fd, dcsize);
int fd, dcsize;

examples: marker_size(3, .060, 0);

dcmarker_size(3, 60);

marker_type

Select marker symbol for subsequent marker primitives
(See Marker Symbol Table).

format: marker_type(fd, marker);
int fd, marker;

example (asterisk): marker_type(3, 2);

move

Update current pen position and move pen to that position.

formats: move2d(fd, wcx, wcy);
int fd;
float wcx, wcy;

dcmove(fd, dcx, dcy);
int fd, dcx, dcy;

examples: move2d(3, 1000.0, 2000.0);

dcmove(3, 535, 464);

perimeter_color

Set color table <index> or RGB color intensity value
<red, green, blue> for subsequent polygon or edged
primitives (rectangle, ellipse, etc.).

formats: perimeter_color_index(fd, index);
int fd, index;

perimeter_color(fd, red, green, blue);
int fd;
float red, green, blue;

COMMAND

FUNCTION

examples (yellow): marker_color_index(3, 3);

marker_color(3, 1.0, 1.0, 0.0);

perimeter_type

Select line type <style> for subsequent polygon and edged primitives (rectangle, ellipse, etc.).

format: perimeter_type(fd, style);
int fd, style;

example (dash): perimeter_type(3, 1);

polygon

Draw a polygon region to be filled and/or edged. The array vertices must immediately follow this command line and be terminated with a ';'.

Note: <clist> is just a string placeholder for array which follows.
< numverts> - number of array vertices
<flags> - 0 -> no move/draw indicators expected in array.
1 -> move(0)/draw(1) indicators must be included in array.

formats: polygon2d(fd, clist, numverts, flags)
int fd;
char "clist";
int numverts, flags;
float array[2 or 3];

dcppolygon(fd, clist, numverts, flags)
int fd;
char "clist";
int numverts, flags;
int array[2 or 3];

examples: polygon2d(3, clist, 4, 1)
2300.0, 5000.0, 0,
2700.0, 5000.0, 1,
2500.0, 4800.0, 0,
2500.0, 5200.0, 1;

dcppolygon(3, clist, 4, 0)
566, 393,
576, 393,
576, 383,
566, 383;

COMMAND

FUNCTION

polyline

Move/draw between specified points. The line array points must immediately follow this command line and be terminated with a ';'.
;

Note: <clist> is just a string placeholder for array which follows.

< numpts > - number of array points

<flags> - 0 -> no move/draw indicators expected in array.

1 -> move(0)/draw(1) indicators must be included in array.

formats: polyline2d(fd, clist, numpts, flags)

```
int fd;  
char "clist";  
int numpts, flags;  
float array[2 or 3];
```

dcpolyline(fd, clist, numpts, flags)

```
int fd;  
char "clist";  
int numpts, flags;  
int array[2 or 3];
```

examples: polyline2d(3, clist, 4, 1)

```
2300.0, 5000.0, 0,  
2700.0, 5000.0, 1,  
2500.0, 4800.0, 0,  
2500.0, 5200.0, 1;
```

dcpolyline(3, clist, 4, 0)

```
566, 393,  
576, 393,  
576, 383,  
566, 383;
```

polymarker

Draw a current marker symbol centered at each point in the array. The array points must immediately follow this command line and be terminated with a ';'.
;

Note: <clist> is just a string placeholder for array which follows.

< numpts > - number of array points

<flags> -0 -> no move/draw indicators expected in array.

1 -> move(0)/draw(1) indicators must be included in array.

COMMAND

FUNCTION

formats: polymarker2d(fd, clist, numpts, flags)

```
int fd;
char "clist";
int numpts, flags;
float array[2 or 3];
```

dcpolymarker(fd, clist, numpts, flags)

```
int fd;
char "clist";
int numpts, flags;
int array[2 or 3];
```

examples: polymarker2d(3, clist, 4, 1)

```
2300.0, 5000.0, 0,
2700.0, 5000.0, 1,
2500.0, 4800.0, 0,
2500.0, 5200.0, 1;
```

dcpolymarker(3, clist, 4, 0)

```
566, 393,
576, 393,
576, 383,
566, 383;
```

rectangle

Draw a rectangular region to be filled and or edged.

formats: rectangle(fd, wxmin, wymin, wxmax, wymax);

```
int fd;
float wxmin, wymin, wxmax, wymax;
```

dcrectangle(fd, dcxmin, dcymin, dcxmax, dcymax);

```
int fd, dcxmin, dcymin, dcxmax, dcymax;
```

examples: rectangle(3, 0.0, 0.0, 5000.0, 5000.0);

dcrectangle(3, 512, 512, 630, 393);

remove_all_names_from_set

Removes all names from current name set. Used in initializing the 'invisibility' filter parameters for turning the segments on or off.

format: remove_all_names_from_set(fd);

```
int fd;
```

example: remove_all_names_from_set(3);

COMMAND

FUNCTION

text

Output a string <string> of characters and display them on screen. Where:

<xform>- 0 -> VDC_TEXT (virtual coord. - 0.0 -> 1.25)
1 -> WORLD_COORDINATE_TEXT
(world coord. - meters)
2 -> TOS_TEXT (don't use)
3 -> ANNOTATION_TEXT (x,y - world coord. /
scale virtual coord. - fixed)

<more> - 0 -> FALSE (no buffering - draw text immediately)

formats: text2d(fd, x, y, string, xform, more);
int fd;
float x, y;
char *string;
int xform, more;

dctext(fd, dcx, dcy, string);
int fd, dcx, dcy;
char *string;

examples: text2d(3, 0.0, 0.0, Text, 3, 0);

dctext(3, 512, 512, Text);

text_color

Set color table <index> or RGB color intensity value <red, green, blue> for subsequent text operations.

formats: text_color_index(fd, index);
int fd, index;

text_color(fd, red, green, blue);
int fd;
float red, green, blue;

examples (cyan): text_color_index(3, 5);
text_color(3, 0.0, 1.0, 1.0);

text_path

Select direction of subsequently drawn text characters. where:

<path> - 0 -> PATH_RIGHT (Along base vector - default)
1 -> PATH_LEFT (180 degrees from base vector)
2 -> PATH_UP (Along the up vector)
3 -> PATH_DOWN (180 degrees from up vector)

COMMAND

FUNCTION

format: text_path(fd, path);

example: text_path(3, 0);

view_window

Define a two-dimensional world coordinate window/viewport in meters.

format: view_window(fd, wcx1, wcy1, wcx2, wcy2);
int fd;
float wcx1, wcy1, wcx2, wcy2;

HP98550A Default RGB Color Table
(4 planes)

Index	Color	Red	Green	Blue
0	black	0.0	0.0	0.0
1	white	1.0	1.0	1.0
2	red	1.0	0.0	0.0
3	yellow	1.0	1.0	0.0
4	green	0.0	1.0	0.0
5	cyan	0.0	1.0	1.0
6	blue	0.0	0.0	1.0
7	magenta	1.0	0.0	1.0
8	10% gray	0.1	0.1	0.1
9	20% gray	0.2	0.2	0.2
10	30% gray	0.3	0.3	0.3
11	40% gray	0.4	0.4	0.4
12	50% gray	0.5	0.5	0.5
13	60% gray	0.6	0.6	0.6
14	70% gray	0.7	0.7	0.7
15	80% gray	0.8	0.8	0.8

Figure H.5. EGS Starbase Colors

Marker Number		
18	Numeral 9	9
17	Numeral 8	8
16	Numeral 7	7
15	Numeral 6	6
14	Numeral 5	5
13	Numeral 4	4
12	Numeral 3	3
11	Numeral 2	2
10	Numeral 1	1
9	Numeral 0	0
8	square_cross	⊠
7	diamond	◇
6	square	□
5	triangle	△
4	"X"	×
3	circle	○
2	star	✱
1	cross	+
0	dot	.

Figure H.6. EGS Starbase Marker Types

Line Number		
0	SOLID
1	DASH
2	DOT
3	DASH_DOT
4	DASH_DOT_DOT
5	LONG_DASH
6	CENTER_DASH
7	CENTER_DASH_DASH

Figure H.7. EGS Starbase Line Types

APPENDIX J

EGS Constants and Prototypes (Include Files)


```

/* sfile.h - include file library for 'sfile_format' editor          */
/* written by : R. H. Meyer      Revision: 1.0   Date: 03-01-1992    */
/* Sandia National Laboratories  Division: 2732                      */

#include <limits.h>          /* include integral type constants */

#ifndef TRUE
#define TRUE      1
#endif
#ifndef FALSE
#define FALSE     0
#endif

#define AppendOn      TRUE
#define AppendOff     FALSE
#define DisplayOn     TRUE
#define DisplayOff    FALSE
#define MaxSegments   100
#define MaxEleSize     10000
#define NoAbbrevArrays FALSE
#define LABEL_UND      38          /* Error Message for 'Label' undefined */
#define END_LABEL      INT_MAX     /* largest possible label number */
#define NO_LABEL       -1         /* No Label for Operation - ... */
                                   /* ... use current pointer position */
#define LINE_LENGTH    81         /* String Array Maximum Line Length */
                                   /* Macro Definitions */
#define chktype(str1,str2,i) (!strcmp((str1),(str2),(i)=strlen(str1)) && \
                              !strcmp((str1),(str2),(i)=strlen(str2)))

                                   /* Function Prototype Definitions */
/* for library - "sfilelb1.c" */
int write_display_list() ;      /* writes a 'Display_List' in "sfile" format */
int read_display_list() ;      /* reads & converts a "sfile" format file */
int read_display_list_segment() ; /* reads & converts a "sfile" segment */
int get_view_data() ;          /* reads & converts AppData I structure */
int set_seg_parms() ;          /* renames segments & filter namesets */
void PrintSegment() ;          /* Print 'ASCII' representation of a Segment */
void RefreshSection() ;        /* redraw a Section of the 'Display_List' */
void DeleteSection() ;         /* delete a Section of the 'Display_List' */
int AddSection() ;             /* add Section to the 'Display_List' */
int ReplaceSection() ;         /* replace Section of 'Display_List' */
int get_ele_type() ;           /* get element type (EGS) from array string */
int parse_str() ;              /* parse command string / return values */
int newline() ;                /* gets another line(s) from Command String */
/* for library - "sfilelb2.c" */
int evl_exc_type() ;           /* evaluate/execute a 'Starbase/Display_List' */
                                   /* command */

```


Appendix J - EGS Constants and Prototypes (Include Files)

```
/* asclib.h - include file library for 'ascii_format' editor          */
/*   written by : R. H. Meyer      Revision: 1.0      Date: 10-18-1993  */
/*   Sandia National Laboratories  Division: 2715          */

#include <float.h>              /* include floating constants    */
#include <math.h>               /* include the Mathj library     */

#define  LINES      200        /* maximum number of lines in buffers */
#ifndef LINE_LENGTH
#define  LINE_LENGTH  81      /* String Array Maximum Line Length */
#endif

/* Macro Definitions */
#define  DEG      *M_PI/180    /* convert degrees to radians */
/* end program flag */
#define  QUIT      (c == 'Q' || c == 'q')

#define  chktype(str1,str2,i) (!strcmp((str1),(str2),(i)=strlen(str1)) && \
                               !strcmp((str1),(str2),(i)=strlen(str2)))

/* Function Prototype Definitions */
/*   for library - "asclib1.c"   */
int open_files() ;             /* gets filenames, checks validity, & opens files */
int get_Input_Buf() ;          /* reads a buffer of text from Input file */
int com_to_asc() ;             /* translates TTR commands to Starbase commands */
/*   for library - "asclib2.c"   */
int read_asc_in() ;            /* read a buffer of text from Input file */
int write_asc_out() ;          /* write a buffer of text to Output file */
int Asc_copy() ;               /* copys a Starbase/Display List 'ascii' buffer */
int form_Hdr() ;               /* formats a Starbase/Display 'ascii' header */
int form_DataI() ;             /* fill an Application Data I structure */
int form_DataII() ;            /* fill an Application Data II structure */
```

```

/* egslib.h - include file library for 'egslib' functions */
/* written by : R. H. Meyer      Revision: 1.0      Date: 10-25-1993 */
/* Sandia National Laboratories  Division: 2715 */

#ifndef TRUE
#define TRUE      1
#endif
#ifndef FALSE
#define FALSE     0
#endif

#define LINE_LENGTH      81      /* String Array Maximum Line Length */
#define FORM_SIZE        25      /* Maximum Entries in Form table */
#define TABS             7       /* set keyboard tab stops */
#define DbufferOn        TRUE     /* sent to "double_buffer" - ON */
#define DbufferOff       FALSE    /* sent to "double_buffer" - OFF */
#define BufferOn         TRUE     /* sent to "buffer_mode" - ON */
#define BufferOff        FALSE    /* sent to "buffer_mode" - OFF */
#define EndOfLine        FALSE    /* sent to "text2d" */
#define AppendText       TRUE     /* sent to "text2d" */
#define Edged            TRUE     /* sent to "interior_style" */
#define NoEdge           FALSE    /* sent to "interior_style" */
#define fm_Dump          TRUE     /* update Font Manager screen - Now */
#define fm_NoDump        FALSE    /* update Font Manager screen -Later */
#define fm_ColorOn       TRUE     /* Font Manager Colors - ON */
#define fm_ColorOff      FALSE    /* Font Manager Colors - OFF */
#define NameMax          1023     /* Maximum name value for filtering */
#define TrailMax         1000     /* Maximum trail length */
#define SMALL_TRACKING_CROSS 3    /* sent to 'echo_type' */
/* Define Function keys */
#define ESC              '\033'   /* escape character */
#define F1               "\033p"  /* softkey f1 */
#define F1_LABEL         " Clear  AER  "
#define F2               "\033q"  /* softkey f2 */
#define F2_LABEL         " Clear  REF  "
#define F3               "\033r"  /* softkey f3 */
#define F3_LABEL         "Nothing"
#define F4               "\033s"  /* softkey f4 */
#define F4_LABEL         " Wind   Run  "
#define F5               "\033t"  /* softkey f5 */
#define F5_LABEL         " Stop   Wind "
#define F6               "\033u"  /* softkey f6 */
#define F6_LABEL         "Nothing"
#define F7               "\033v"  /* softkey f7 */
#define F7_LABEL         "Nothing"
#define F8               "\033w"  /* softkey f8 */
#define F8_LABEL         " Quit   "
/* Define Special keys */
#define SELECT           "\033&P" /* Select key */
#define UP_ARROW         "\033A"  /* Arrow keys */
#define DOWN_ARROW       "\033B"
#define RIGHT_ARROW      "\033C"
#define LEFT_ARROW       "\033D"
#define HOME             "\033h"  /* Home Up key */

```

Appendix J - EGS Constants and Prototypes (Include Files)

```

#define SHIFT_HOME      "\033F"      /* Home Down key */
#define CLEAR_DISPLAY   "\033J"      /* Clear Display key */
#define CLEAR_LINE      "\033K"      /* Clear Line key */
#define INSERT_LINE     "\033L"      /* Insert Line key */
#define DELETE_LINE     "\033M"      /* Delete Line key */
#define DELETE_CHAR     "\033P"      /* Delete Character key */
#define INSERT_CHAR     "\033Q"      /* Insert Character key */
#define ROLL_UP         "\033S"      /* Shift Up-arrow key */
#define ROLL_DOWN       "\033T"      /* Shift Down-arrow key */
#define NEXT            "\033U"      /* Next key */
#define PREVIOUS        "\033V"      /* Previous key */
#define CNTRL_A         '\01'        /* Control-A character */
#define CNTRL_B         '\02'        /* Control-B character */

/* Define colors - 16 entry color table */
#define Black           0
#define White           1
#define Red             2
#define Yellow          3
#define Green           4
#define Cyan            5
#define Blue            6
#define Magenta         7
#define Gray10          8
#define Gray20          9
#define Gray30         10
#define Gray40         11
#define Gray50         12
#define Gray60         13
#define Gray70         14
#define Gray80         15

/* Define Symbols */
#define SYMBOL_SIZE     .01          /* marker size */
#define BLINK_RATE      200          /* marker blink rate(milliseconds) */
#define TRAIL_SYMBOL    3            /* marker - circle (3) */
#define LABEL_SYMBOL    5            /* marker - triangle (5) */
#define LABEL_COLOR     Magenta
#define HACK_SYMBOL     7            /* marker - diamond (7) */
#define ARADAR_SYMBOL   4            /* marker - cross (4) */
#define ARADAR_COLOR    Red
#define BRADAR_SYMBOL   8            /* marker - square with cross (4) */
#define BRADAR_COLOR    Yellow
#define IN1_SYMBOL      1            /* marker - plus (1) */
#define IN1_COLOR       Green
#define IN2_SYMBOL      2            /* marker - asterisk (2) */
#define IN2_COLOR       Blue
#define D_LINE_STYLE    DASH_DOT_DOT /* draw line - style & color */
#define D_LINE_COLOR    Green

/* Macro's definitions */
/* end program flag */
#define QUIT             (c == 'Q' || c == 'q')
/* "choice_string" buffer flag */
#define QUEUE_EMPTY      (status == EMPTY_OVERFLOW || \
                          status == EMPTY_NO_OVERFLOW)
/* Valid A & B Radar flags */

```

```

#define VALID_A_INDEX ((Read_Data_Block.A_Index > -1) && \
                      (Read_Data_Block.A_Index < 8))
#define VALID_B_INDEX ((Read_Data_Block.B_Index > -1) && \
                      (Read_Data_Block.B_Index < 8))
/* bit-manipulation macros */
#define Wrdsz (int)(sizeof(int) * 8)
#define Arraymax (int)(NameMax / Wrdsz + 1)
#define Setbit(array,bit) array[(bit) / Wrdsz] |= 1 << ((bit) % Wrdsz)
#define Clearbit(array,bit) array[(bit) / Wrdsz] &= ~(1 << ((bit) % Wrdsz))
#define Clearallbits(array) { int ii; \
                             for (ii = 0; ii < Arraymax; ii++) \
                             array[ii] = 0; \
                             }
/* Relative time compare Macro */
#define Chk_time(t1,t0,val) ((float)((t1)-(t0)) > (val))

/* define the Graphics Cursor (user-defined) */
static unsigned char cursor[64] = { /* Font Cursor pattern */
    3, 3, 3, 2, 2, 2, 2, 3,
    3, 3, 3, 3, 3, 3, 2, 3,
    3, 3, 3, 3, 3, 3, 2, 3,
    3, 3, 3, 3, 3, 2, 3, 3,
    3, 3, 3, 3, 2, 3, 3, 3,
    3, 3, 3, 3, 2, 3, 3, 3,
    3, 3, 3, 3, 2, 3, 3, 3,
    3, 3, 3, 3, 2, 3, 3, 3 } ;

/* define Application Data/ Shared Memory structures */
struct file_seg_chr { /* file location structure */
    char path[22] ; /* Path for filename */
    char filename[16] ; /* filename where segment stored */
} ;

struct map_chr { /* map (view) structure */
    char swch[5] ; /* Control Panel switch */
    char view_name[5] ; /* Control View Name */
    struct file_seg_chr file ; /* this filename/ location */
    char xoffset[9] ; /* map zero offset from... */
    char yoffset[9] ; /* range zero (x, y, z) */
    char zoffset[9] ;
    char zoom[9] ; /* map magnification factor */
    char orientation[5] ; /* map azimuth - north */
} ;

struct AppData_chr {
    char remark[81] ; /* Remarks & Comments */
    char author[21] ; /* Author */
    char date[11] ; /* Date - Last revised */
    char rev[5] ; /* Current revision number */
    struct map_chr mp_data ; /* map structure */
} ;

/* define other structure types */
typedef struct Pstn { /* position structure in TTR coordinates */
    float x ;

```

```

    float      y ;
    float      z ;
    float      h ;
} POSITION, *POSITN ;

typedef struct lbl {                /* label structure */
    POSITION    pos ;                /* label location */
    char       time[16] ;           /* label time string */
    char       alt[16] ;            /* label altitude string */
    struct lbl *next ;              /* pointer to next label in list */
} lbl, *lb ;

typedef struct hck {                /* hack structure */
    POSITION    pos ;                /* hack location */
    struct hck *next ;              /* pointer to next hack in list */
} Hack, *hk ;

typedef struct ln {                 /* draw line structure */
    POSITION    pos ;                /* point location */
    int        md ;                 /* move/draw flag */
    struct ln  *next ;              /* pointer to next hack in list */
} LINE, *LN ;

/* Function prototype Definitions */

/* for library - 'egslib1.c' */
void init_graph_devices() ;
void init_wndow() ;
void init_Font_Manager() ;
void Set_Softkeys() ;
void get_key_command() ;
unsigned char *save_popup() ;
void restore_popup() ;
int choice_string() ;
void ProcessKey() ;
void close_graph_devices() ;
void Set_CTM() ;
int Identity2D() ;
int Scale2D() ;
int Translate2D() ;
int Rotate2D() ;
int Concat2D() ;
void Form_33() ;
void Form_32() ;
/* for library - 'egslib2.c' */
void Plot_Radar_Sym() ;
void UpDate_Trail() ;
void Draw_Line() ;
void add_line_lst() ;
void free_line_list() ;
void Get_RefIn() ;
void Display_AER() ;
void Clear_AER() ;
void Display_REF() ;
void Clear_REF() ;

```

```
/* for library - 'egslib3.c' */
void Remove_Com_Shm() ;
void SetCom_defaults() ;
int Checkstring() ;
int parse_lat_lng() ;
void Write_Label() ;
void UpDate_Labels() ;
void Clear_Labels() ;
void add_label_lst() ;
void free_label_list() ;
void Write_Hack() ;
void UpDate_Hacks() ;
void Clear_Hacks() ;
void add_hack_lst() ;
void free_hack_list() ;
void Plot_Symbol() ;
/* for library - 'frmlib1.c' */
void form3() ;
void form3_fill() ;
void form4() ;
void form4_fill() ;
```

Appendix J – EGS Constants and Prototypes (Include Files)

```

/* ttrcoord.h - header file for "ttrcoord.c" & "rngcoord.c" library      */
/*   written by: R. H. Meyer      Revision: 1.0      Date: 03-03-1994      */
/*   Sandia National Laboratories  Division: 2715      */
#include <math.h>                  /* need Math library */

#ifndef M_PI
#define M_PI (double)3.141592654
#endif
#ifndef DEG
#define DEG *M_PI/180.0          /* convert degrees to radians */
#endif
#define RAD *180.0/M_PI          /* convert radians to degrees */
#define FEET *(double).3048      /* convert feet to meters */
#define METERS *(double)3.280839895013123 /* convert meters to feet */
#define NMILES *(double)1852.    /* convert nautical miles to meters */
#define N (double)0.0
#define MDCL (double)-16.5       /* TTR magnetic declination (degrees) */
/* Constants for NAD 27 */
/* TTR origin - East translation (meters) */
#define E027 (double)-2268291.512085
/* TTR origin - translation (meters) */
#define F027 (double)-4504316.024198
/* TTR origin - North translation(meters) */
#define G027 (double)3891538.461524
/* TTR geodial height (meters) */
#define N27 (double)18.2
/* Ellipsoid radius - Major-axis (meters) */
#define A27 (double)6378206.400
/* Ellipsoid radius - Minor-axis (meters) */
#define B27 (double)6356583.800
/* Flattening ratio */
#define F27 (double)0.0033900753
/* First Eccentricity Squared */
#define ESQRD27 (double)0.0067686580
/* Second Eccentricity Squared */
#define EPSQRD27 (double)0.0068147849
/* TTR Range Zero latitude, longitude, & height(msl) - NAD27 */
#define ZLAT27 "37 50'36.71085"
#define ZLNG27 "-116 43'44.37431"
#define ZHGT27 (double)0.0
/* Constants for WGS 72 */
/* TTR origin - East translation (meters) */
#define E072 (double)-2268316.403361
/* TTR origin - translation (meters) */
#define F072 (double)-4504159.317779
/* TTR origin - North translation(meters) */
#define G072 (double)3891716.575929
/* TTR geodial height (meters) */
#define N72 (double)28.64
/* Ellipsoid radius - Major-axis (meters) */
#define A72 (double)6378135.000
/* Ellipsoid radius - Minor-axis (meters) */
#define B72 (double)6356750.520
/* Flattening ratio */

```

```

#define F72 (double)0.0033527795
/* First Eccentricity Squared */
#define ESQRD72 (double)0.0066943178
/* Second Eccentricity Squared */
#define EPSQRD72 (double)0.0067394337
/* TTR Range Zero latitude, longitude, & height(msl) - WGS72 */
#define ZLAT72 "37 50'36.35853"
#define ZLNG72 "-116 43'48.16634"
#define ZHGT72 (double)0.0
/* Constants for WGS 84 */
/* TTR origin - East translation (meters) */
/* #define E084 (double)-0000000.000000 */
/* TTR origin - translation (meters) */
/* #define F084 (double)-0000000.000000 */
/* TTR origin - North translation(meters) */
/* #define G084 (double)0000000.000000 */
/* TTR geodial height (meters) */
#define N84 (double)23.33
/* Ellipsoid radius - Major-axis (meters) */
#define A84 (double)6378137.000
/* Ellipsoid radius - Minor-axis (meters) */
#define B84 (double)6356752.314
/* Flattening ratio */
#define F84 (double)0.0033528107
/* First Eccentricity Squared */
#define ESQRD84 (double)0.0066943800
/* Second Eccentricity Squared */
#define EPSQRD84 (double)0.0067394967
/* Datum Translations (meters) */
#define DX84 (double)-7.9
#define DY84 (double)159.7
#define DZ84 (double)175.5
/* TTR Range Zero latitude, longitude, & height(msl) - WGS84 */
#define ZLAT84 "37 50'36.503"
#define ZLNG84 "-116 43'47.600"
#define ZHGT84 (double)0.0
/* function prototypes - ttrcoord.c */
double sdms_deg(char *) ;
void deg_sdms(double, char *) ;
void rotat1(double,double,double,double,double,double*,double*,double*) ;
void rotat2(double,double,double,double*,double*,double*,double,double) ;
void trng_earth(double*,double*,double*,double,double,double) ;
void tearth_rng(double*,double*,double*,double,double,double) ;
void xyz(double,double,double,double*,double*,double*) ;
void llh(double,double,double,double*,double*,double*) ;
void trng_earth72(double*,double*,double*,double,double,double) ;
void tearth_rng72(double*,double*,double*,double,double,double) ;
void xyz72(double,double,double,double*,double*,double*) ;
void llh72(double,double,double,double*,double*,double*) ;
void wgsnad(double,double,double,int,double*,double*,double*) ;
/* function prototypes - rngcoord.c */
void xyz_rae(double,double,double,double*,double*,double*) ;
void rae_xyz(double,double,double,double*,double*,double*) ;
double conv_magn(double) ;

```


Appendix J – EGS Constants and Prototypes (Include Files)

```
double conv_truen(double) ;
void rotate_cl(double,double,double,double,double,double*,double*) ;
double time_to_go(double,double) ;
/* function prototype - molod.c */
void molod(double,double,double,double,double,double,double,double,double,
double,double,double*,double*,double*) ;
```

APPENDIX K

EGS/WayPoints Coordinate System Equations

Method and Equations originally developed on Tektronix 4081 system by D. F. Anderson (2719).

WayPoint Parameters (Given in WayPoint File):

Previous Point - $\langle x_0, y_0, z_0, r_0 \rangle$

Current Point - $\langle x_1, y_1, z_1, r_1 \rangle$

Next Point - $\langle x_2, y_2, z_2, r_2 \rangle$

x, y, z = meters

r_0, r_1, r_2 = turn radius (nautical miles)

radius = $r_1 * 1852$ (meters) /* Convert to meters */

$\pi = 3.141592654$

I. (headg) Solve for Line Equations: $y = mx + b$

$$\text{(slope)} \quad m_0 = dy / dx = \frac{y_1 - y_0}{x_1 - x_0}; \quad m_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\text{(angle)} \quad a_0 = \arctan \left[\frac{y_1 - y_0}{x_1 - x_0} \right] \text{(radians)}$$

$$a_1 = \arctan \left[\frac{y_2 - y_1}{x_2 - x_1} \right] - \pi \quad \text{(radians)}$$

$$\text{(heading)} \quad \varnothing_0 = 90 - a_0 * 180 / \pi \text{ (degree)}$$

north

$$\varnothing_1 = 90 - a_1 * 180 / \pi \text{ (degree)}$$

$$\text{(intercept)} \quad b_0 = y_0 - m_0 * x_0; \quad b_1 = y_1 - m_1 * x_1$$

II. Find Arc & Distances

$$\text{(Arc)} \quad \lambda = (|\varnothing_1 - \varnothing_0| / 2) * \pi / 180 \quad \text{(radians)}$$

$$\text{(Distance)} \quad tdist = \text{radius} * \tan(\lambda) \quad \text{(meters)}$$

Point(s) $\langle x_d, y_d \rangle$ which lie on each line and are a distance (tdist) from the current WayPoint $\langle x_1, y_1 \rangle$.

$$\begin{aligned} x_{d0} &= x_1 + tdist * \cos(a_0 - \pi) \\ y_{d0} &= y_1 + tdist * \sin(a_0 - \pi) \end{aligned} \quad \text{(meters)}$$

$$\begin{aligned} x_{d1} &= x_1 + tdist * \cos(a_1) \\ y_{d1} &= y_1 + tdist * \sin(a_1) \end{aligned} \quad \text{(meters)}$$

III. Find Orthogonal Line Segments (Perpendicular)

(Negative Reciprocal Slopes)

$$im0 = -1 / m0 = -\frac{x1 - x0}{y1 - y0}; \quad im1 = -1 / m1 = -\frac{x2 - x1}{y2 - y1}$$

$$b0 = yd0 - im0 * xd0 = yd0 + (1/m0) * xd0$$

$$b1 = yd1 - im1 * xd1 = yd1 + (1/m1) * xd1$$

IV. Calculate the Intersection of these two lines.

$$yc = im0 * xc + b0$$

$$yc = im1 * xc + b1$$

$$xc = \frac{b1 - b0}{im0 - im1}; \quad yc = \frac{im0 * b1 - im1 * b0}{im0 - im1}$$

V. (circ) Find the Arc Start and Stop Points.

$$(start) \quad \theta 1 = \arctan \left[\frac{yd0 - yc}{xd0 - xc} \right] * 180 / \pi \quad (\text{degrees})$$

$$start = \theta 1 * \pi / 180 \quad (\text{radians})$$

$$(stop) \quad \theta 2 = \arctan \left[\frac{yd1 - yc}{xd1 - xc} \right] * 180 / \pi \quad (\text{degrees})$$

$$stop = \theta 2 * \pi / 180 \quad (\text{radians})$$

VI. Therefore the Starbase arc drawing primitive is:

$$\text{arc}(3, \text{radius}, \text{radius}, xc, yc, start, stop);$$

Figure K.1. WayPoint Arc Drawing Algorithm

APPENDIX L

'Egs_master' and 'Command_proc' Flow Charts

q

Figure L.1. 'EGS_master' Flow Chart (main)

b

Figure L.1. 'EGS_master' Flow Chart (main)

Figure L.2. 'set_default_values' Flow Chart (function)

Figure L.3. 'Init_Shm_Memory" Flow Chart (function)

Figure L.4. 'Command_proc' Flow Chart (main)

Figure L.5. 'check_cmd' Flow Chart (function)

Figure L.5. 'check_cmd' Flow Chart (function)

Figure L.5. 'check_cmd' Flow Chart (function)

d

Figure L.5. 'check_cmd' Flow Chart (function)

APPENDIX M

'Graphics_proc' Flow Charts

a

Figure M.1. 'Graphics_proc' Flow Chart (main)

b

Figure M.1. 'Graphics_proc' Flow Chart (main)

Figure M.1. 'Graphics_proc' Flow Chart (main)

d

Figure M.1. 'Graphics_proc' Flow Chart (main)

e

Figure M.1. 'Graphics_proc' Flow Chart (main)

f

Figure M.1. 'Graphics_proc' Flow Chart (main)

9

Figure M.1. 'Graphics_proc' Flow Chart (main)

Figure M.1. 'Graphics_proc' Flow Chart (main)

Figure M.2. 'Set_Window_Parms' Flow Chart (function)

Figure M.2. 'Set_Window_Parms' Flow Chart (function)

a

Figure M.3. 'Refresh_Screen' Flow Chart (function)

Figure M.3. 'Refresh_Screen' Flow Chart (function)

Figure M.3. 'Refresh_Screen' Flow Chart (function)

d

Figure M.3. 'Refresh_Screen' Flow Chart (function)

e

Figure M.3. 'Refresh_Screen' Flow Chart (function)

Figure M.3. 'Refresh_Screen' Flow Chart (function)

Figure M.4. 'Draw_Static_windows' Flow Chart (function)

a

Figure M.5. 'Check_Events' Flow Chart (function)

b

Figure M.5. 'Check_Events' Flow Chart (function)

C

Figure M.5. 'Check_Events' Flow Chart (function)

d

Figure M.5. 'Check_Events' Flow Chart (function)

e

Figure M.5. 'Check_Events' Flow Chart (function)

Figure M.5. 'Check_Events' Flow Chart (function)

Figure M.6. 'Time_Search_Value' Flow Chart (function)

Figure M.7. 'Plot_Radar_Sym' Flow Chart (function)

b

Figure M.7. 'Plot_Radar_Sym' Flow Chart (function)

C

Figure M.7. 'Plot_Radar_Sym' Flow Chart (function)

d

Figure M.7. 'Plot_Radar_Sym' Flow Chart (function)

e

Figure M.7. 'Plot_Radar_Sym' Flow Chart (function)

Figure M.8. 'Display_AER' Flow Chart (function)

Figure M.8. 'Display_AER' Flow Chart (function)

Figure M.8. 'Display_AER' Flow Chart (function)

Figure M.9. 'Display_REF' Flow Chart (function)

Figure M.10. 'Draw_Line' Flow Chart (function)

Figure M.10. 'Draw_Line' Flow Chart (function)

Figure M.10. 'Draw_Line' Flow Chart (function)

Figure M.10. 'Draw_Line' Flow Chart (function)

e

Figure M.10. 'Draw_Line' Flow Chart (function)

Figure M.10. 'Draw_Line' Flow Chart (function)

Figure M.11. 'form3' Flow Chart (function)

Figure M.11. 'form3' Flow Chart (function)

Figure M.11. 'form3' Flow Chart (function)

Figure M.12. 'form3_fill' Flow Chart (function)

b

Figure M.12. 'form3_fill' Flow Chart (function)

Figure M.12. 'form3_fill' Flow Chart (function)

Figure M.13. 'Wind_Run' Flow Chart (function)

b

Figure M.13. 'Wind_Run' Flow Chart (function)

APPENDIX N

EGS/TTR Save/Restore/Transfer Scripts

Figure N.1. EGS Directory Hierarchy

I. Save (Backup) Script

The 'save_EGS' backup script saves the current contents of the '/users/egs' directory to the HP9145 Tape Drive. As an option, the user can provide a directory {MASK} to selectively save specific directory sub-trees.

1. Make sure the HP9145 Cartridge Tape Drive is Ready and Write enabled.
 - a. **Turn on** the HP9145 Tape Drive if needed.
 - b. Turn the **SAFE** knob on the Cartridge away from the Safe Position.
 - c. **Load** a blank 1/4" Cartridge Tape (32 Track) into the Drive Unit and wait for the '**Ready**' light to come on.

Warning: Make sure that the 'Write Protect' light is not illuminated.

2. At the 'login' prompt on the HP370 EGS System type the following.

```
login: egs <CR>
password: ***** <CR>
```

Note: In the integrated Setup 'X-window' environment, make the login terminal window the Input focus window (or create a 'New Window' from the Root Window).

3. Run the 'save_EGS' script to backup the EGS directories to the Cartridge Tape.
 - a. In response to the shell prompt(\$) type one of the following.

Complete EGS Backup

\$ save_EGS <CR>

or Partial EGS Backup (see Figure N.1)

\$ save_EGS {MASK} <CR>

Where {MASK} is the partial directory path(s) to be backed up to tape. Note only this directory path and its sub-trees are saved to the tape.

More than one directory path name may be specified in this {MASK}. All specified path names must be able to fit on one command line and each path name must be separated by a space from the other path names.

I. **Save (Backup) Script** (continued)

Example: Save the '/users/egs/data' directory and all the Project Directories.

```
$ save_EGS ./data <CR>
```

Example: Save the '/users/egs/graphics' EGS directory executable files only.

```
$ save_EGS ./graphics <CR>
```

Warning: Note each call to 'save_EGS' starts writing to the cartridge tape at the beginning of the tape. Multiple calls to the 'save_EGS' script cannot be used to store consecutive directories. Each call will over-write the previous tape contents.

b. Answer the 'save_EGS' prompt as follows:

Is the Tape Drive Ready [Write Enabled]? (y or n, default: y) y <CR>

Respond 'y' or 'CR' if the HP9145 Tape Drive is Ready (Step #1) for writing and you want to continue.

'n' to abort the backup operation.

c. The selected EGS directory files are now copied to the Tape Unit and the file names are displayed on the terminal screen.

d. When the 'save_EGS' backup process is finished the following message will be displayed.

Unload the Tape Drive [Set the Write Protect Tab] !

4. Exit the HP370 user's terminal window by typing the following.

```
$ exit <CR>
```

5. Unload the HP9145 Cartridge Tape Drive and set the Write Enable Tab.

a. **UnLoad** the 1/4" Cartridge Tape from the Drive Unit and wait for all the lights to go out.

b. Press the **Eject** Button to remove the Tape Cartridge from the Unit.

c. Turn the **SAFE** knob on the Cartridge to the Safe Position.

d. **Turn off** the the HP9145 Tape Drive if desired.

II. Restore Script

The 'restore_EGS' script copies the current contents of the HP9145 Tape Drive to the corresponding '/users/egs/*' directory. As an option, the user can provide a directory {MASK} to selectively restore specific directory sub-trees.

1. Make sure the HP9145 Cartridge Tape Drive is Ready and Write Protected.
 - a. **Turn on** the the HP9145 Tape Drive if needed.
 - b. Turn the **SAFE** knob on the Cartridge to the Safe Position.
 - c. **Load** the 1/4" Cartridge Tape into the Drive Unit and wait for the 'Ready' light to come on.

Warning: Make sure that the 'Write Protect' light is illuminated.

2. At the 'login' prompt on the HP370 EGS System type the following.

```
login: egs <CR>
password: ***** <CR>
```

Note: In the integrated Setup 'X-window' environment, make the login terminal window the Input focus window (or create a 'New Window' from the Root Window).

Warning: Executable files cannot be restored from the Tape if they are currently active processes. This is especially a problem in the integrated Setup environment and with the daemon program 'pollpio'. All executable files must be shutdown before they can be updated.

3. Run the 'restore_EGS' script to restore the EGS directories from the Cartridge Tape.
 - a. In response to the shell prompt(\$) type one of the following.

Complete EGS Restore (Everything that is on the Tape)

```
$ restore_EGS <CR>
```

or Partial EGS Restore (see Figure N.1)

```
$ restore_EGS {MASK} <CR>
```

Where {MASK} is the partial directory path(s) to be restored from tape. Note only this directory path and its sub-trees are copied from the tape. Sub-directories are created if needed.

II. Restore Script (continued)

More than one directory path name may be specified in this {MASK}. All specified path names must be able to fit on one command line and each pathname must be separated by a space from the other path names.

Example: Restore the '/users/egs/data' directory and all the Project Directories.

```
$ restore_EGS ./data <CR>
```

Example: Restore the '/users/egs/graphics' EGS directory executable Files only (See above warning).

```
$ restore_EGS ./graphics <CR>
```

Warning: The specified {MASK} must have been previously saved on the cartridge tape or nothing will be restored.

- b. Answer the 'restore_EGS' prompt as follows:

Is the Tape Drive Ready [Write Protected]? (y or n, default: y) y <CR>

Respond 'y' or 'CR' if the HP9145 Tape Drive is Ready (Step #1) for reading and you want to continue.

'n' to abort the restore operation.

- c. The selected EGS directory files are now copied from the Tape Unit and the file names are displayed on the terminal screen.
- d. When the 'restore_EGS' process is finished the following message will be displayed.

Unload the Tape Drive !

4. Exit the HP370 user's terminal window by typing the following.

```
$ exit <CR>
```

5. Unload the HP9145 Cartridge Tape Drive.

- a. **UnLoad** the 1/4" Cartridge Tape from the Drive Unit and wait for all the lights to go out.
- b. Press the **Eject** Button to remove the Tape Cartridge from the Unit.
- c. **Turn off** the the HP9145 Tape Drive if desired.

III. Put (Send) Transfer Script

The 'put_EGS' script copies the current contents of the '/users/egs' directory to a Remote System over the Ethernet LAN connection. As an option, the user can provide a directory {MASK} to selectively copy specific directory sub-trees.

1. Make sure the Remote System is operational and the Ethernet LAN connection is working.
2. At the 'login' prompt on the Local HP370 EGS System type the following.

```
login: egs <CR>
password: ***** <CR>
```

Note: In the integrated Setup 'X-window' environment, make the login terminal window the Input focus window (or create a 'New Window' from the Root Window).

Warning: Executable files cannot be copied to a Remote Host if the file is currently an active process on the Remote System. This is especially a problem in the integrated Setup environment and with the daemon program 'pollgpiio'. All executable files must be shutdown before they can be updated on the Remote system.

3. Run the 'put_EGS' script to copy the EGS directories to the Remote Host System.
 - a. In response to the shell prompt(\$) type one of the following.

Complete EGS Copy

```
$ put_EGS <CR>
```

or Partial EGS Copy (see Figure N.1)

```
$ put_EGS {MASK} <CR>
```

Where {MASK} is the partial directory path(s) to be copied to the Remote System. Note only this directory path and its sub- trees are copied; sub-directories are created (one level) if needed.

More than one directory path name may be specified in this {MASK}. All specified path names must be able to fit on one command line and each path name must be separated by a space from the other path names.

III. Put (Send) Transfer Script (continued)

- b. Answer the 'put_EGS' prompt(s) as follows:

All EGS Copies

Enter the Remote Host Name: **RemoteName** <CR>

Respond with the Remote System Name (or alias) as defined in the '/etc/hosts' and '\$HOME/.netrc' files.

Complete EGS Copy (only)

Do you really want to copy the complete '/users/egs' directory ? (y or n, default: n) **y**
<CR>

Respond 'y' if you want to continue.

'n' or 'CR' to abort the copy operation.

Example: Copy the 'ProjectTest' directory to a Remote System named "hp2".

```
$ put_EGS ProjectTest <CR>
```

Enter the Remote Host Name: hp2 <CR>

Example: Copy the '/users/egs/data' directory and all the Project Directories to a Remote System named "hp1".

```
$ put_EGS ./data <CR>
```

Enter the Remote Host Name: hp1 <CR>

Example: Copy the '/users/egs/graphics' EGS directory executable Files only (See above warning) to a Remote System named "hp3".

```
$ put_EGS ./graphics <CR>
```

Enter the Remote Host Name: hp3 <CR>

- c. The selected EGS directory files are now copied to the Remote EGS System from the Local EGS System and the file names are displayed on the terminal screen as they are copied.

III. Put (Send) Transfer Script (continued)

- d. When the 'put_EGS' process is finished the following message will be displayed.

nnn Goodbye.

where: nnn is some sequence number.

4. Exit the HP370 user's terminal window by typing the following.

\$ exit <CR>

IV. Get (Receive) Transfer Script

The 'get_EGS' script copies the current contents of the '/users/egs' directory from a remote System over the Ethernet LAN connection. As an option, the user can provide a directory {MASK} to selectively copy specific directory sub-trees.

1. Make sure the Remote System is operational and the Ethernet LAN connection is working.
2. At the 'login' prompt on the Local HP370 EGS System type the following.

login: egs <CR>
password: ***** <CR>

Note: In the integrated Setup 'X-window' environment, make the login terminal window the Input focus window (or create a 'New Window' from the Root Window).

Warning: Executable files cannot be copied from a Remote Host if the file is currently an active process on the Local System. This is especially a problem in the integrated Setup environment and with the daemon program 'pollpio'. All executable files must be shutdown before they can be updated on the Local system.

3. Run the 'get_EGS' script to copy the EGS directories from the Remote Host System.

- a. In response to the shell prompt(\$) type one of the following.

Complete EGS Copy

\$ get_EGS <CR>

or Partial EGS Copy (see Figure N.1)

\$ get_EGS {MASK} <CR>

Where {MASK} is the partial directory path(s) to be copied from the Remote System. Note only this directory path and its sub-trees are copied; sub-directories are created (one level) if needed.

More than one directory path name may be specified in this {MASK}. All specified path names must be able to fit on one command line and each path name must be separated by a space from the other path names.

- b. Answer the 'get_EGS' prompt(s) as follows:

All EGS Copies

Enter the Remote Host Name: **RemoteName** <CR>

Respond with the Remote System Name (or alias) are defined in the '/etc/hosts' and '\$HOME/.netrc' files.

Complete EGS Copy (only)

Do you really want to copy the complete '/users/egs' directory ? (y or n, default: n) y
<CR>

Respond 'y' if you want to continue.

'n' or 'CR' to abort the copy operation.

Example: Copy the 'ProjectTest' directory from a Remote System named "hp2".

\$ get_EGS ProjectTest <CR>

Enter the Remote Host Name: hp2 <CR>

Example: Copy the '/users/egs/data' directory and all the Project Directories from a Remote System named "hp1".

\$ get_EGS ./data <CR>

Enter the Remote Host Name: hp1 <CR>

Example: Copy the '/users/egs/graphics' EGS directory executable Files only (See above warning) from a Remote System named "hp3".

\$ get_EGS ./graphics <CR>

Enter the Remote Host Name: hp3 <CR>

- c. The selected EGS directory files are now copied from the Remote EGS System to the Local EGS System and the file names are displayed on the terminal screen as they are copied.

IV. **Get (Receive) Transfer Script** (continued)

- d. When the 'get_EGS' process is finished the following message will be displayed.

nnn Goodbye.

where: nnn is some sequence number.

4. Exit the HP370 user's terminal window by typing the following.

\$ exit <CR>

DISTRIBUTION:

1	MS0507	K. G. McCaughey, 2700
1	MS0577	T. J. Baca, 2741
5	0577	R. H. Meyer, 2741
1	MS0555	J. R. García, 2742
2	0555	K. C. Bauhs, 2742
1	MS0555	D. B. Davis, 2732
1	MS1392	L. W. Lathrop, 2719
4	1392	D. F. Anderson, 2719
1	1392	J. A. Enlow, 2719
3	1392	TTR Department File, 2719
1	MS1392	J. G. Dykes, 2719-1
1	1392	W. D. Love, 2719-1
1	1392	R. C. Sherwood, 2719-1
1	1392	M. E. Spsychala, 2719-1
1	MS0865	S. L. Stronach, 2753
1	MS9018	Central Technical Files, 8523-2
5	0899	Technical Library, 13414
1	0619	Technical Publications, 13416
10	0100	Document Processing, 7613-2
		For DOE/OSTI