

RATTLE COLUMBUS LABS OH F/6 13/13
STRUCTURAL ANALYSIS VIA GENERALIZED INTERACTIVE GRAPHICS - STAG--ETC(U)
SEP 79 L E HULBERT, N D GHADIALI, F N DEOBOT F33615-76-C-3125
AFFDL-TR-79-3074-VOL-3 NL

AFEDL-TR-79-3074-VOL-3 NL

1 OF 2

AD
SUNSHINE 4/2/2

Abstract

AFFDL-TR-79-3074
Volume III

AD A089332

**STRUCTURAL ANALYSIS VIA GENERALIZED
INTERACTIVE GRAPHICS
STAGING**
Volume III — System Manual

*L. E. HULBERT
C. P. SCOFIELD*

*BATTELLE COLUMBUS LABORATORIES
505 KING AVENUE
COLUMBUS, OHIO 43201*

SEPTEMBER 1979

TECHNICAL REPORT AFFDL-TR-79-3074, Volume III
Final Report for Period June 1976 — September 1979

REC
SEP 13 1980

Approved for public release; distribution unlimited.

DDC FILE COPY

AIR FORCE FLIGHT DYNAMICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

80 9 22 237

NOTICE

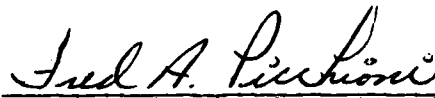
When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.



Bernard H. Groomes
Project Engineer



Frederick A. Picchioni, Lt Col, USAF
Chief, Analysis & Optimization Branch
Structures & Dynamics Division

FOR THE COMMANDER



Ralph L. Kuster Jr., Col. USAF
Chief, Structures & Dynamics Division

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/FIBR, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER AFFDL-TR-79-3074-V61-31V	2. GOVT ACCESSION NO. AD-A089382	3. RECIPIENT'S CATALOG NUMBER 4	
4. TITLE (and Subtitle) STRUCTURAL ANALYSIS VIA GENERALIZED INTERACTIVE GRAPHICS - STAGING. VOLUME III. SYSTEM MANUAL.		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL - FINAL 28 JUNE 1976 - SEPT 1979	
6. AUTHOR(s) L. E. HULBERT, N. D. GHADIALI, F. N. DEOBOT		7. PERFORMING ORG. REPORT NUMBER A089382	
8. CONTRACT OR GRANT NUMBER(s) F-33615-76-C-3125		9. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 12154	
10. PERFORMING ORGANIZATION NAME AND ADDRESS BATTELLE, COLUMBUS LABORATORIES 505 KING AVENUE COLUMBUS OHIO 43201		11. CONTROLLING OFFICE NAME AND ADDRESS AIR FORCE FLIGHT DYNAMICS LABORATORY (FBRA) WRIGHT-PATTERSON AIR FORCE BASE OHIO 45433	
12. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (16) 2411		13. REPORT DATE SEP 1979	
14. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED		15. NUMBER OF PAGES 158	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		16. SECURITY CLASS (of this report) UNCLASSIFIED	
18. SUPPLEMENTARY NOTES		17. DECLASSIFICATION DOWNGRADING SCHEDULE	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) INTERACTIVE GRAPHICS FINITE ELEMENT MODELS STRUCTURAL ANALYSIS MESH GENERATION COMPUTER AIDED DESIGN COMPUTER AIDED ANALYSIS			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) STAGING (STRUCTURAL ANALYSIS via Generalized Interactive Graphics) has been developed to give engineers an interactive graphics system for constructing and studying finite element models and for reviewing the results of a finite element analysis. Volume I consists of a user's guide giving detailed step-by-step instructions in how to use STAGING.			

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FOREWARD

This final report was prepared by the Columbus Laboratories of Battelle Memorial Institute, Columbus, Ohio, for the Structures and Dynamics Division, Air Force Flight Dynamics Laboratory, Wright-Patterson Air Force Base, Ohio. The work was performed under Contract No. F-33615-75-C-3125, which was initiated under Project No. 2401, "Structures and Dynamics", Task No. 02, "Design and Analysis Methods for Aerospace Vehicle Structures". Initially, Mr. L. Bernier (FBR) was the AFFDL project engineer for this effort, after which Mr. B.H. Groomes (FBR) was assigned the responsibility.

STAGING, as described in this report, represents a three-year combined Air Force-Navy effort, with specific support and contributions from Dr. Charles P. Porier, Chief, Scientific Systems Analysis Branch, and Computer Center, Wright-Patterson Air Force Base, Ohio, Messers. James M. McKee and Michael E. Golden, Computation Mathematics and Statistics Department, Code 1844, Mr. Paul Mayer and Miss Jane A. Figula, Structures Department, Code 1730.5, The David W. Taylor Naval Ship Research and Development Center, Bethesda, Maryland. The technical graphics expertise of these government researchers are gratefully acknowledged.

The report consists of four volumes. Volume I, "Final Summary Report", presents an overview of the capabilities of the STAGING (STructural Analysis via Generalized Interactive Graphics) system. Volume II, "Users Guide", gives detailed instructions on how to use STAGING for finite element analysis. Volume III, "System Manual", describes the internal structure of STAGING and details procedures for installation and Maintenance of the System on CDC CYBER and 6000 series mainframe computers. Volume IV, "Appendices to the System Manual", includes lists of STAGING procedures, loader directives and cross-referenced tables of all entry names that occur in STAGING.

The program manager of this development was Dr. L. E. Hulbert of the Transportation and Structures Department. He was supported by N. D. Ghadiali of the same department and by a number of specialists from the Computer, Information Systems, and Education Department including:

E. Edwards	K. Cadmus
D. Kasik	C. Scofield
W. Young	F. Drobot

Kevin Cadmus was a major contributor to the preparation of this volume.

The work reported herein was conducted during the period of June 28, 1976 through June 1979. Some work on STAGING was carried out under contract F33615.

The present report was submitted for publication in June, 1979.

TABLE OF CONTENTS

1. INSTALLATION

- 1.1 STAGING Files
- 1.2 Loading the STAGING Absolute
- 1.3 Loading the Menu Generator Absolute
- 1.4 Loading the Material Property Data Base Management System
- 1.5 Loading Conversion Routine Absolutes

2. 2.1 Cataloged Procedures

- 2.1.1 Introduction
- 2.1.2 Low-Level Procedures
- 2.1.3 Running STAGING
- 2.1.4 Maintenance of STAGING Libraries
- 2.1.5 Menu Generation
- 2.1.6 Maintenance of Material Property Data Base
- 2.1.7 Installation of WPAFB
- 2.1.8 YIELDFILE Procedures
- 2.1.9 Procedure Listings
- 2.2 Low-Level System Interface and Utility Routines
- 2.3 Data Handler
- 2.4 STAGING Menu Generation and Management
 - 2.4.1 Introduction
 - 2.4.2 Using the Command Tree Generator
 - 2.4.3 Menu Data Base
 - 2.4.4 Menu Generation Software
 - 2.4.5 Processing the Command Tree in STAGING Driver
 - 2.4.6 Textual Input
- 2.5 STAGING Model Data Base
 - 2.5.1 Introduction
 - 2.5.2 STAGING Model Data Base Format
 - 2.5.3 Change List
 - 2.5.4 In-Core Tables and Arrays
 - 2.5.5 Model Data Base Handler Routines
- 2.6 INTERTEK Interactive Graphics System
 - 2.6.1 Introduction
 - 2.6.2 INTERTEK Picture Manipulation
 - 2.6.3 INTERTEK Pick Processing
 - 2.6.4 INTERTEK Software Overview
- 2.7 STAGING Model Graphics
 - 2.7.1 Medium-Level Graphics Routines
 - 2.7.2 Construction of Model Display
 - 2.7.3 Results Displays
 - 2.7.4 Modifying the Picture
 - 2.7.5 Editing the Model
 - 2.7.6 Graphical Input
- 2.8 STAGING Material Property Data Base
 - 2.8.1 Introduction
 - 2.8.2 MPDB Modification Procedure
 - 2.8.3 MPDB Format

Accession For

- 2.9 STAGING Code
 - 2.9.1 Overview of STAGING Code
 - 2.9.2 Program Library Maintenance
 - 2.9.3 Segmentation Strategy

LIST OF TABLES

- 1.1 - STAGING Files
- 2.4.1 - Menu Generation Software
- 2.5.2 - Element Attributes
- 2.5.3 - Element Type Values
- 2.5.4 - Node Attributes
- 2.7.1 - View Control Array

LIST OF FIGURES

FIGURES

- 2.4.1 - Sample Menu Generator Input Deck
- 2.4.2 - Sample Menu Generator Output (1)
- 2.4.3 - Sample Menu Generator Output (2)
- 2.4.4 - Triplet for Text Editing
- 2.4.5 - More Than Button Picking
- 2.4.6 - Tree Schematic
- 2.4.7 - Menu Data Base Bead Format
- 2.5.1 - Overall Structure of Beads
- 2.5.2 - Format of Data Base Beads
- 2.5.3 - Table Bead Format
- 2.5.4 - Change Bead Format
- 2.5.5 - Change File Header Bead
- 2.5.6 - Filling ELNOD
- 2.5.7 - ILKUP Array with Index INLKUP
- 2.5.8 - Attribute Names
- 2.5.9 - Attribute Index into Bead
- 2.5.10 - Paging Scheme
- 2.5.11 - Information Flow
- 2.6.1 - Display File Construction
- 2.6.2 - DAE Format
- 2.6.3 - Subfile Format
- 2.6.4 - Item Format
- 2.6.5 - Display Item Graphics Instruction Formats
- 2.6.6 - IBUF Format
- 2.6.7 - Extension Format
- 2.6.8 - Category Table
- 2.6.9 - Queue Construction
- 2.6.10 - Wait Queue Entities
- 2.6.11 - Keyboard Event Table
- 2.6.12 - Rectangle Check
- 2.6.13 - Distance Test
- 2.6.14 - Test Pick
- 2.7.1 - 3D View Control

1. INSTALLATION

The installation of the STAGING system involves loading files to disk from the STAGING System Tape and creating several program absolutes. In the following sections are descriptions of the files on the System Tape and the procedures for accessing the files and loading the absolutes.

The STAGING System Tape is a 9-track, 1600 bpi tape. All files are copied to the tape using COPYBF. The source program libraries are copied to tape as source decks in UPDATE input format. The relocatable libraries are loaded as sequential libraries created via the RANTOSEQ command to EDITIB.

1.1.1 STAGING Files

A table of the files in the STAGING system is given in Table 1.1. When the STAGING absolute (STAGINGABS) is loaded, subprograms are retrieved from five sources:

1. STAGINGLIB
2. SUPPORTLIB
3. INTERTEKLIB
4. MENUTABLE
5. Standard NOS/BE system libraries

STAGINGLIB contains all subroutines on UPDATE program library STAGINGPL. (The compiled binaries are on STAGINGBIN.) These routines include the STAGING main program ("DRIVER"), the routines called by DRIVER for menu management, and the subroutines associated with picks of menu buttons.

SUPPORTLIB contains data base manipulation routines, conversion routines, and miscellaneous routines for permanent file manipulation, error recovery, and XY-plot graphics support. The source code for these SUPPORTLIB routines is maintained on UPDATE program libraries and binaries:

1. DATABASEPL,DATABASEBIN
2. CONVERSIONPL,CONVERSIONBIN
3. PERMFILEPL,PERMFILEBIN
4. RECOVERYPL,RECOVERYBIN
5. XYGRAPHPL,XYGRAPHBIN

TABLE 1.1. STAGING FILES

File Name	Format	Contents
PROCFIL	CCL	STAGING procedure file
PROFIL	Univ. of Wash. CL	STAGING procedure file
STAGINGPL	UPDATE	High level STAGING routines including DRIVER
INTERTEKPL	UPDATE	INTERTEK interactive graphics package
CONVERSIONPL	UPDATE	Conversion routines
DATABASEPL	UPDATE	Data base manipulation routines including Data Handler
XYGRAPHPL	UPDATE	XY plot support package
PERMFILEPL	UPDATE	Permanent file manipulation routines
RECOVERYPL	UPDATE	ERROR recovery package
MENUGENPL	UPDATE	Menu generation subsystem
STAGINGBIN	LGO	Result of compiling all routines on STAGINGPL
INTERTEKBIN	LGO	Result of compiling all routines on INTERTEKPL
CONVERSIONBIN	LGO	Result of compiling all routines on CONVERSIONPL
DATABASEBIN	LGO	Result of compiling all routines on DATABASEPL
XYGRAPHBIN	LGO	Result of compiling all routines on XYGRAPHPL
PERMFILEBIN	LGO	Result of compiling all routines on PERMFILEPL
RECOVERYBIN	LGO	Result of compiling all routines on RECOVERYPL
MENUGENBIN	LGO	Result of compiling all routines on MENUGENPL

TABLE 1.1. STAGING FILES (CONTINUED)

File Name	Format	Contents
STAGINGLIB	EDITLIB	From STAGINGBIN
INTERTEKLIB	EDITLIB	From INTERTEKBIN
SUPPORTLIB	EDITLIB	From CONVERSIONBIN, DATABASEBIN, XYGRAPHBIN, PERMFILEBIN, AND RECOVERYBIN
STAGINGABS	ABSOLUTE	STAGING system
MENUGENABS	ABSOLUTE	Menu generator
MPDBNEWABS	ABSOLUTE	MPDB creator
MPDBADDABS	ABSOLUTE	MPDB modifier
MPDB	Data Handler	Material Property Data Base
EXECMENUSOURCE	Text	Executive menu definition
GLOBALMENUSOURCE	Text	Global menu definition
DISPLAYMENUSOURCE	Text	Display and Edit menu definition
PREMENUSOURCE	Text	Preprocessor menu definition
POSTMENUSOURCE	Text	Postprocessor menu definition
EXECMENUDRIVER	FTN	Switching table and situation dependence function
GLOBALMENUDRIVER	FTN	Switching table and situation dependence function
DISPLAYMENUDRIVER	FTN	Switching table and situation dependence function
PREMENUDRIVER	FTN	Switching table and situation dependence function
POSTMENUDRIVER	FTN	Switching table and situation dependence function

TABLE 1.1. STAGING FILES (CONTINUED)

File Name	Format	Contents
EXECMENU	Data Handler	Run-time menu data base
GLOBALMENU	Data Handler	Run-time menu data base
DISPLAYMENU	Data Handler	Run-time menu data base
PREMENU	Data Handler	Run-time menu data base
POSTMENU	Data Handler	Run-time menu data base
MENUTABLE	LGO	Result of compiling all above MENUMDRIVERS

INTERTEKLIB contains the routines on UPDATE program library INTERTEKPL (binaries on INTERTEKBIN). These routines support all interactive graphics used in STAGING (see section 2.3). INTERTEKLIB, like STAGINGLIB and SUPPORTLIB, are in standard NOS/BE library format suitable for maintenance via EDITLIB.

The file MENUTABLE contains 2 binaries for each menu (module) in STAGING. The first binary is the assembled menu driver containing calls to each subroutine callable from the menu. It acts as a simple switching table allowing a menu button pick to activate the appropriate subroutine. The second binary is the situation dependent decision function (see section 2.5). Both routines are created by the menu generator system.

The STAGING menu generator absolute (MENUGENABS) is created by loading the subprograms from UPDATE program library MENUGENPL. (The binaries are on MENUGENBIN). SUPPORTLIB must be declared as a library.

The source input to the STAGING menu generator are maintained on files with file names having format:

prefixMENUSOURCE

where the prefix is EXEC, GLOBAL, DISPLAY, PRE, or POST. The source code output from the menu generator is cataloged as:

prefixMENUDRIVER

and the menu data base itself is cataloged as

prefixMENU.

The maintenance of the STAGING Material Property Data Base (MPDB) requires two programs. Absolute MPDBNEWABS (source MPDBNEW) is used to create an empty Material Property Data Base in Data Handler format. Absolute MPDBADDABS (source MPDBADD) is used to revise the contents of the Material Property Data Base. The source code for these two programs is maintained on STAGINGPL. File MPDB is the Material Property Data Base. The absolutes for these programs are generated by the following commands:

ATTACH(PROFIL,ID=STAGING3)

BEGIN,MPDBGENABS,TYPE=NEW.

BEGIN,MPDBGENABS,TYPE=ADD.

See Appendix A for listings of these procedures.

The conversion routine source code delivered with the STAGING system are:

NASTRAN: NASCON1
 NASCON2
 NASCON3
 FASTOP: SOPCON1
 SOPCON2
 SOPCON3
 FOPCON1
 FOPCON3
 AXISOL: AXICON1
 AXICON2
 AXICON3
 HONDO: HND01
 HND03
 DOASIS: DOASC01
 DOASC03
 ADINA: ADICON1

The source and binaries are maintained on CONVERSIONPL and CONVERSIONBIN respectively. SUPPORTLIB is to be declared a library prior to loading a conversion routine absolute.

1.2 Loading the STAGING Absolute

To load the STAGING absolute:

```
ATTACH(PROFIL,ID=STAGING3)
BEGIN,STAGINGGEN,MAP=PART.
```

For installations with CCL (Cyber Control Language) substitute file PROCFIL for PROFIL. The MAP parameter controls the segloader output listing. MAP=ON gives a more extensive listing; MAP=OFF suppresses the listing. STSEG requires the availability of the following files:

```
PROFIL
MENUTABLE
SEGDIR
STAGINGLIB
SUPPORTLIB
INTERTEKLIB
```

See Appendix A for a listing of procedure STAGINGGEN.

1.3 Loading the Menu Generator Absolute

To load the Menu Generator absolute:

```
ATTACH(PROFIL,ID=STAGING3)
```

```
BEGIN,MENUGENABS.
```

For installations with CCL (Cyber Control Language), substitute file PROCFIL for PROFIL.

MENUGENABS requires the availability of the following files:

MENUGENBIN

SUPPORTLIB

See Appendix A for a listing of procedure MENUGENABS.

1.4 Loading the Material Property Data Base Management System

There are two absolutes, MPDBNEWABS and MPDBADDABS, in the Material Property Data Base Management System. To load them:

```
ATTACH(PROFIL,ID=STAGING3)
```

```
BEGIN,MPDBGENABS,TYPE=NEW.
```

```
RETAIN.
```

```
BEGIN,MPDBGENABS,TYPE=ADD.
```

For installations with CCL (Cyber Control Language), substitute file PROCFIL for PROFIL.

MPDBGENABS requires the availability of the following files:

SUPPORTLIB

STAGINGPL

See Appendix A for a listing of procedure MPDBGENABS.

1.5 Loading a Conversion Routine

Creation of the absolute of a conversion routine is straightforward. Only library SUPPORTLIB is needed. As an example, the following control cards can be used to create the absolute of NASTRAN conversion program 1:

```
ATTACH(OLDPL,CONVERSIONPL,ID=STAGING3)
```

UPDATE(Q,I=INPUT)

input deck: *ID N1

*C NASCON1

FTN(I)

ATTACH(LIB,SUPPORTLIB)

LIBRARY(LIB)

LOAD(LGO)

NOGO(ABS)

CATALOG(ABS,NASCON1ABS,ID=STAGING3)

Some of the conversion routines (e.g. for SOP) require large field lengths for loading and will have to be done in batch mode.

2. SYSTEM MAINTENANCE

The maintainer of the STAGING system should become familiar with the various aspects of STAGING in a particular order. The person should first become familiar with using the various modules of the system. Next comes understanding the nature of each file in the system and the purpose and operation of each cataloged procedure. Before modifying any code in the STAGING system, the maintainer should become aware of the segmentation strategy used in STAGING and the unusual problems involved in working in a segmentation environment. Some familiarity with the Data Handler is a necessary prerequisite for understanding most of the STAGING code. Then the following elements of STAGING can be studied independently:

- a) STAGING Menu Generation and Management
- b) STAGING Model Data Base and Conversion Routines
- c) INTERTEK Interactive Graphics System.

Any modifications to the GPRIME system should be coordinated with David Taylor NSRDC.

2.1 STAGING Cataloged Procedures

2.1.1 Introduction

The procedures required to utilize and maintain the STAGING system are maintained on file PROFIL. These procedures have been rewritten into CCL format so that they can be used at Wright Patterson Air Force Base (WPAFB). The corresponding file is PROCFIL.

PROFIL contains procedures for maintaining program libraries and auxiliary files in the STAGING system. STAGING has the capability of interactively executing a procedure on. Before such a procedure is executed, STAGING is swapped out of memory and, at the completion of the called procedure, STAGING is swapped back into memory with all files intact. Execution then begins where it left off. The only procedure currently yielded to from STAGING is XGPRIME which initiates execution of the GPRIME system.

Complete listings of all procedures on PROFIL and PROCFIL are given in Appendix A.

2.1.2 Low Level Procedures

The most common step within a cataloged procedure is to ATTACH, CATALOG, or REQUEST a permanent file. ATTACH and REQUEST require a preliminary RETURN of the local file name for safety. CATALOG is safeguarded against failure due to too many existing cycles--the lowest cycle is PURGED and the CATALOG retried. For these reasons it was useful to write three low level procedures to perform the details of these permanent file functions. These low level procedures (A, C, and R) are on PROFIL. Since a permanent file name can be 40 characters long but procedure parameters are limited to 10 characters, there are four permanent file name parameters passed to these procedures. Execution of the procedure concatenates the parameters to form the full permanent file name.

2.1.3 Running STAGING

To execute the STAGING system, simply execute procedure STAGING on file PROFIL. First ATTACH,PROFIL,ID=STAGING3 and then BEGIN,STAGING. (At WPAFB the user should ATTACH,PROCFIL,ID=STAGING3 and BEGIN,STAGING.)

2.1.4 Maintenance of STAGING Libraries

The following procedures are used to maintain STAGING libraries:

UPDATE

NEWLIB

SUPPORTLIB

UPDATE is used to make any changes to source code on any of the STAGING program libraries. The call is

BEGIN,UPDATE,PROFIL,prefix

where prefix is one of the following:

STAGING

INTERTEK

MENUGEN

DATABASE

CONVERSION

XYGRAPH

RECOVERY

PERMFILE

The changes are to have been placed on file prefixIN before running this procedure. A new program library and binary file are automatically cataloged and the input file emptied. If prefix=STAGING or INTERTEK, then the corresponding random library is automatically created (see procedure NEWLIB below). If prefix=MENUGEN, then a new menu generator absolute is created (see procedure MENUGENABS discussed in Section 1.3). If prefix=

DATABASE, CONVERSION, XYGRAPH, RECOVERY, or PERMFILE, then a new SUPPORTLIB is created (see procedure SUPPORTLIB below). Note that the STAGING absolute is never regenerated by a call to procedure UPDATE (see Section 1.2 for discussion of loading the STAGING absolute).

NEWLIB is used by procedure UPDATE to create a new STAGINGLIB or INTERTEKLIB from the appropriate binary file (STAGINGBIN or INTERTEKBIN).

SUPPORTLIB is used by procedure UPDATE to create a new SUPPORTLIB. Input consists of binary files DATABASEBIN, CONVERSIONBIN, XYGRAPHBIN, RECOVERYBIN, and PERMFILEBIN.

2.1.5 Menu Generation

STAGING menus are created by execution of procedure MENUGEN. Input to MENUGEN consists of the situation dependent routine (FTN source code) followed by the coded description of the menu to be generated. This coded description includes menu hierarchy position, the menu button name, the subroutines associated with the button, the resultant movement in the menu tree, and special action indicators. It may also include an option parameter which indicates that the button is to be displayed only if the situation dependent function subroutine returns a value of 1.

The menu generation absolute is maintained on file MENUGENABS. If any changes are needed in the menu generation system, procedure MENUGENABS may be used to create a new cycle of MENUGENABS.

During execution of MENUGEN, three inquiries will be displayed at the terminal:

- a. DO YOU WANT AN ECHO OF INPUT LINES?
- b. TREE GENERATED. WOULD YOU LIKE IT PRINTED?
- c. HOW ABOUT A FULL FILE DUMP?

Normally the answers should be NO, YES, NO, respectively. The file dump option is only useful for debugging the menu generation system. Any printed output is placed on file OUTPUT. Input to MENUGENABS resides on file prefixMENUSOURCE where prefix is EXEC, GLOBAL, DISPLAY, PRE, or POST. The menu database is CATALOGed as file prefixMENU. The subroutine table

and situation dependent function are in UPDATE input format and are CATALOGED on file prefixMENUDRIVER. They are compiled in procedure MENUTABLE to create new load file MENUTABLE.

2.1.6 Maintenance of Material Property Data Base

Two procedures on PROFIL affect the STAGING Material Property Data Base. Procedure MPDBNEW creates an empty MPDB. Procedure MPDBADD processes additions and revisions to the MPDB. Section 2.4 gives a detailed description of the preparation of input for procedure MPDBADD.

2.1.7 Installation at WPAFB

Installation of the STAGING absolute system at WPAFB is accomplished by creating a system tape containing all necessary files and loading the tape to disk at WPAFB. Once the files are loaded at WPAFB, it is necessary to create absolutes for STAGING, GPRIME, Menu Generator, Material Property Data Base System, and conversion routines as described in Section 1.2 through 1.6.

If any difficulties arise due to differences in operating systems, it is necessary to recreate the PL's, BIN's, and LIB's by recompiling the source code supplied on the system tape in UPDATE input deck format.

2.1.8 YELDFILE Procedures

Certain button picks cause STAGING to be swapped out of core and a call made to execute a procedure on PROFIL. At completion of the procedure, STAGING is swapped back into core and begins execution where it left off. The parameters to the PROFIL procedures are specified in the default field of the bead in the menu data base corresponding to the button pick.

2.1.9 Procedure Listings

Each procedure in the STAGING procedure files is listed in two formats. First is the University of Washington Control Language format used at Battelle and until recently at WPAFB. Second is Cyber Control Language format recently adopted at AFFDL Computer Center. Both formats for all procedures on PROFIL and PROCFIL are listed in Appendix A.

2.2 Low-Level System Interface and Utility Routines

Much of the day-to-day maintenance of the STAGING system depends upon the operating system environment. The code in the package should transfer readily as the FTN compiler is upgraded because the code is compiled in OPT=1. The COMPASS file manipulation routines in COPYFL use the options of system routine CPC and PP routine CIO. Neither routine has changed from a user's point of view in years and should remain constant to STAGING as operating systems are changed.

The routines that will cause most problems during operating-system transitions are Battelle developed routines including:

PERMFIL, RETURN, REQUEST: These routines are FORTRAN-callable permanent file utilities. RETURN and REQUEST have been quite stable. PERMFIL has had more difficulties. Error processing can be unstable because CDC has a tendency to change error numbers around. The only specific error tested in STAGING is 10. (note floating point error numbers) which indicates that the file attached is not in the system. The version of PERMFIL provided allows cataloging a file without specifying an account (the account the user logged-in with is used).

The Battelle PERMFIL routines provide all the permanent file capabilities needed to ATTACH, CATALOG, EXTEND, REQUEST, etc. a file on the CDC system.

IPFUT: Puts one more level between the programmer and PERMFIL by performing the requested function on array PFNAME. Special case: RENAME puts an infinite retention period on the file to save the user the trouble of entering that information.

The following routines are in the system to increase the aesthetics of a display or to make life a bit easier for the programmer:

COPYFL: Copy the left adjusted trailing zero local file in parameter 1 to the left adjusted trailing zero local file in parameter 2. The file is copied until EOI is reached. The files are rewound before and after copying if parameter3 is omitted or zero. If

parameter3 is non-zero, the file positions are unaffected before and after the copy.

IBEAUT: Take the number input, determine if it is floating or integer, encode it, and return it left justified with trail blanks and the number of significant characters.

ICRACK: Parse a packed character array into a lower and upper bound for a range of values.

ICCHR: Count the number of characters in a Hollerith string before the trailing blanks in array parameter 1. If parameter 2 is omitted, parameter 1 is assumed to be one word long, else the number of words of parameter 1 to scan.

IDINF: Each item corresponding to a bead in a display contains the bead address of the data base item. When the item is picked, the ID is returned and IDINF reconstructs and returns the bead address from ID (2) and ID (3) in IDENTs.

INDATA: Read from an internally connected file and crack into fields based on "=" and "," characters. Each field can be of arbitrary length (over 10 characters). The results are split and returned as character strings, character counts, and pointers in three arrays in common block /INPUT/.

MBEAD: Form the ID block out of a bead address. Returned in ID (1) to ID (3) in IDENTs.

MV: Move a bit string from one word to another. Will cross word boundaries and up or down arrays.

Entry point

MBV: Move character string.

UNPACK: Return one character at a time from a packed character string.

One of the STAGING functions is to pass control to a separate independent task or main program. This is similar to a FORTRAN program calling a subroutine. The subroutine does its job and control is passed

to the calling program at the first executable statement after the call. When the called main program has performed its task, the calling program is restarted at the first executable statement following the call. Two pieces are needed to make this scheme work. The first piece is a swapper (called IYIELD) that restarts STAGING at exactly the same spot in the application. The second piece is a routine (called EXEC) which passes control to a predefined procedure file and makes sure that the STAGING files are intact after execution.

IYIELD is the heart of the swapping process. It writes a full core image, the exchange jump package, and a small bootstrap program onto a local file (ZZZZZX), passes control to a stack of control cards, and restarts STAGING by executing the bootstrap program.

The user is allowed to pass up to 640 characters of control cards to the routine. The code works on a start, issue, end philosophy to save rewrites of the checkpoint file. Thus a control card record is begun with a CALL IYIELDS, added to with a call to IYIELD (string, no. characters in string), and executed by a call to IYIELDE(1). (The 1 indicates that the caller is to be restarted upon return). The IYIELDE call issues a request to the PP program IAP with a pointer to the card stack. If the program is to be restarted, the bootstrap program (ZZZZZX) is executed.

The control cards to be issued have the following constraints: (1) do not call a MUJ job (EDITOR), (2) do not try to LOGOUT or, (3) do not issue a call to another program that uses IAP to issue control cards. Item (3) poses the only real restriction, and that restriction only prevents automatic restart. A second call to IAP eliminates the stack of control cards, i.e. eliminates the restart location. Thus certain utilities cannot be used. Similarly, a routine which calls IYIELD from a program previously yielded to will cause problems because IYIELD calls IAP.

To make YIELDING as easy to use as possible, all the problems in dealing with files were eliminated. The user need not worry about what files are in use, random or sequential, connected or disconnected, open or closed. The EXECC routine returns all proper files before the procedure specified is executed. Upon return, those files are restored and the yielding program can continue. Not that the display file is also saved and restored. The format for calling EXECC is:

CALL EXECC (String)

Where string is a procedure name followed by optional parameters and terminated by a period. The string need not be delimited. Thus

"XYZ". and

"PROC3,PFN=GOLDI,ID=LOCKS."

are both valid inputs. The string must not be greater than 50 characters long.

EXECC then operates on the string to produce

BEGIN,procname,procfile,parameter list.

The procedure name must be on procedure file PROFIL,ID=STAGING3. Additional control cards can be executed after procname is executed. The routine in procname can return another stack of control cards to the yielding program by writing the list on TAPE30.

A shorthand for routine EXECC is provided by routine EXEC for rapid code integration into the executive menu. EXEC looks at the "D=" fields in the menu source to construct the parameter in EXECC. The importance of this approach is that STAGING does not need to be reloaded to integrate a new code.

2.3 Data Handler

The Data Handler used in the STAGING system is an implementation of a virtual memory manager for the storage of large data bases. It was designed to be efficient for interactive applications. It provides for easy deletion, addition, and modification of data in general digraph structures. It is used in STAGING for the manipulation of data in the model data base, the display file data base, the menu data bases, and the material property data base. The data in these structures are contained in variable length "beads" each of which is assigned a unique bead address. Beads contain pointer fields which contain bead addresses of other beads logically related to the given bead.

Beads are grouped into logical blocks for efficient access. Recently accessed blocks are retained in-core as long as possible to minimize the transfers to and from mass storage.

The Data Handler can manage up to 8 data base files at one time. Pointers to in-core block buffers are kept in common block DMTBL which must be loaded in-core at a lower address than any of the buffer arrays. (Proper operation of the Data Handler requires that no files array index be negative). The file names are not to appear in the FTN PROGRAM statement.

Data Handler Subroutines

DMINIT-- establish data file and in-core block space for a data base
CALL DMINIT(ifile, iblk, iblk1, npru, lfile)
where ifile = file name (1 to 7 characters, left
justified, zero filled)
 iblk = dimensioned array for I/O buffer
 iblk1 = length of iblk buffer array
 npru = number of PRU's in a file block
 (1 PRU = 64 words)
 lfile = logical file 0 to 7 (default 0)
DMGTBD-- allocate space on the file for a specified length bead,
 initialize to zero, and return the assigned bead address

CALL DMGTBD (nwords, ibead, lfile)

where nwords = number of 60 bit words to allocate
ibead = returned bead address
lfile = logical file 0 to 7 (default 0)

DMSET-- store data into a bead

CALL DMSET (icomp,ibead,val)

where icomp = a code defining the placement within the bead
ibead = bead address
val = value or array of values (right-justified)
to be stored in the bead

DMGET-- retrieve a value from a bead

CALL DMGET (icomp,ibead,val)

where icomp = a code defining the field within a bead
ibead = bead address
val = value or array of values to receive
the data from the bead

DMRLBD-- release (delete) a bead from the data base

CALL DMRLBD (ibead)

where ibead = bead address of bead to be returned
to free storage space (returned as zero)

DMFLSH-- update the mass storage file using the in-core blocks; then
close the data base file and free the buffer space.

CALL DMFLSH (lfile)

where lfile = logical file number from 0 to 7 (default 0)

2.4 Staging Menu Generation and Management

2.4.1 Introduction

The STAGING system is constructed around a general purpose command tree generator system. This system allows a system designer to express the intended user interaction in a free format language. The menu generation system processes this language and creates a menu file in Data Handler format needed during program execution. It also generates the source of subroutines needed to determine which buttons are displayable and the appropriate subroutine to be associated with each button.

2.4.2. Using the Command Tree Generator

Each module in the STAGING system requires a command tree of menus. A tree generation program, external to the STAGING system, performs this task. It processes an input file that is composed of 80-column card images.

The card image file is split into three sections. Section 1 consists of two cards telling the generator: (a) the logical unit on which the menu data handler file is to be generated (3 if the menus are global, else 2) (b) the name of the subroutine driver table and the user supplied situation decision function.

The second section is initiated by a *BEGIN (in column 1) and terminated by a *END. This section contains the source code of the situation dependent decision function which determines whether a menu item should appear on the screen. The STAGING program retrieves the "option" field from the menu data base. This field contains an integer which is passed as a parameter to the situation dependent decision function and determines which test is to be performed. For example, the ERASE SCREEN button in Display and Edit tests IF (NUMDAD .EQ. 0). The function must return nonzero for a displayed button, else zero. See Figure 2.4.1 for a concrete example.

The decision functions and subroutine driver table must be compiled and loaded with the rest of the application. The menu generator

```

EXTABLEIEXDEC
*BEGIN
    FUNCTION IEXDEC(KEY)
*CALL DATBAS
*CALL ATTRIB
    IEXDEC = 0
    IF(KEY.EQ.1)GO TO 1
    IF(KEY.EQ.2)GO TO 2
    GO TO 100

C
C     IS THERE AN ACTIVE DATA BASE
C
1 IF(FILES(IUNITD+1).NE.0.) 100,101

C
C     IS THERE AN ACTIVE DATA BASE **AND**
C     IS STRESS IN DATA BASE FOR POSTPROCESSING
C
2 IF(FILES(IUNITD+1).EQ.0.) GO TO 101
    L=LENATT(3)
    IF(L.EQ.0)GO TO 101
    DO 20 I=1,L
        N=IGATT(I,3)
        N=NAMFNM(N,3)
        N=SHIFT(N,-18).AND. .NOT. MASK(24)
        IF(N .EQ. 6R N STR)GO TO 100
20 CONTINUE
    GO TO 101
100 IEXDEC = 1
101 RETURN
    END

*END
1,  L=-1,S=FILATT,A=2,D="PROLOG",
P="ENTER DATA BASE FILE. IF NEW FILE OR ATTACHED AS TAPE0, TYPE SPACE.",
2,  L = -1, P = "CHOOSE MAJOR MODULE.",
2,  N="PREPROCESSORS",      A=2, T=5,
3,  L = -1, P = "CHOOSE PREPROCESSOR",
3,  N="GPRIME",             A=2, T=5,
4,  L=-1, P="NEW OR RESTART.",
4,  N="GPRIME-NEW",         A=-1,T=5,S=EXEC,
    D="XGPRIME,IN",D="ITIAL.",
4,  N="GPRIME-RESTART",     A=-1,T=5,S=EXEC,
    D="XGPRIME,RE",D="START.",
3,  N = "OTHER PREPROCESSORS", A = 2, T = 5, S = PRINIT,
4,  N="RETURN",             A=-1,T=5,
3,  N = "RETURN", A = -1, T = 5,
2,  N="DISPLAY AND EDIT",   A=2, T=5,S=DEINIT,O=1,
2,  N="POST PROCESSORS",    A=2, T=5,S=PPINIT,O=2,
2,  N="RETURN",             A=-1,T=5,

```

FIGURE 2.4.1 - SAMPLE MENU GENERATOR INPUT DECK

provides these routines in an UPDATE input format which is used in procedure MENUTABLE when the new menus are to be loaded into a new STAGING absolute. This allows the source of the decision function to query variables in COMDECK's on STAGINGPL.

The third section of the source is the actual set of menu cards. The only fixed input data field is the level specification. The input mechanism processes all 80 columns of a card image. No special continuation marks are needed. The scan will continue until the next level number is encountered. All fields (including the level number) are delimited by commas. Blanks are insignificant except when enclosed in quotation marks ("). These fields describe the light button text pick, prompting messages, and the action to take in the tree. Each field is started by a key letter and an '=' sign. The key letters and meanings are:

N = "phrase"--

The light button text to be placed on the screen. A practical limit for phrase is 25 characters.

L = integer number--

The location on the screen in the y direction at which the button name will be placed. The integer number is a position from 1 to 20 that will cause relative text placement in increments of 100 rasters. L is generally unused because the generator assigns the x and y positions automatically. Special case: If L = -1, the bead is assumed to contain prompting information only.

P = "message"--

The prompting message to appear on the screen when the menu is displayed. Flagged by L = -1. The message has a practical limit of 70 characters.

T = integer number--

The button pick type. 5 is a button pick, 6 is a special action pick. T = 6 is unused in this version of the STAGING system.

S = subroutine name--

The subroutine to be called when the button name is picked.

A = integer number--

Action or movement in the tree to be taken after the subroutine has been called. The action can be overridden by the subroutine

(which is handy if an error has occurred). The subroutine need only set variable IER in COMDECK ERROR to the desired value. The values are interpreted as:

- N: Go up the tree (toward level 1) N levels. If the tree is at level 17 and A = -2, picking the button in the N field will return to level 15 of the tree. Used invariably with a RETURN button.
- 1: Continue processing picks. The application may need string, single, or parameter picks. A = 1 yields a call to the subroutine in the 'S =' field for each new pick fetched (but not for the button pick). A call to the subroutine is made to allow the application to clean up any loose ends. Pick information is returned in COMDECK IDENTs.
- 2: Go down one level in the tree.
- 4: Stay on the same level.
- 5: Stay on the same level and redisplay the menu. This action is used as an error override to reinitiate text editing properly when problems occur.

D = anything--

Default values with ten characters maximum. Up to 50 D values can be assigned to any one name. Currently, only A = 1 actions require presence of the D field. If A = 1, D will specify the action to be taken in the tree after the subroutine has been called.

O = + integer--

Option in situation decision function. If omitted, the button will always appear, else the test (based on the integer parameter) will be executed to determine if the button is displayable in this situation.

These action fields provide all the information the menu generator needs to produce the proper files to drive any application. If an error occurs, the generator ignores the error and continues as well as possible.

The fields required to define a bead (light button and associated data) depend on its type: button name or prompting message. The button name bead needs the N =, A =, and T = fields to function properly (a subroutine call is by no means mandatory). A prompting message bead needs only P = and L = -1.

An example of the input and resultant output is given in Figures 2.4.1, 2.4.2, and 2.4.3.

Two situations arise which may cause confusion in the tree. First, text editing is a problem because the user must be able to specify a subroutine to process this input and the action to be taken in the tree. The solution is given in Figure 2.4.4. The programmer adds the S and A fields to a prompting message bead on the same level RETURN would naturally occupy. See Section 2.4.6 for a detailed description of STAGING text input facilities.

The second situation requires a bit more programmer input. The program has the need to query and process string, single, or parameter picks on occasion. The programmer must set up the pick type he is expecting in COMDECK CATS which contains five categories (ICAT) and meanings (MEAN) for the categories. The meaning can be any of the legitimate actions in INTERTEK. The reason for this specification is to speed up auxiliary pick processing by telling the DRIVER what to expect next. This saves much time because the driver does not have to retrieve single or parameter picks if a string pick is expected. The menu bead for the A = field is exemplified in Figure 2.4.5. Because the processing subroutine is called only for the non-button picks, some mechanism is needed to proceed through the tree. The D field supplies the tree action to be taken, while the A = 1 field contains pick processing strategy.

Output from the generator is two files. The first is the data handler file containing the menu tree. The second is the source COMPASS subprogram that contains the subroutine table of all 'S =' parameters in the tree and the situation decision function. These two subprograms must be loaded with the driver and user subroutines when loading the STAGING absolute. Menu generation and loading are done automatically in procedures MENUGEN and STAGINGGEN on PROFIL.

SUBROUTINE TABLE AND DECISION ROUTINES--

```

*AF IN
*DECK EXTABLE
      IDENT EXTABLE
      ENTRY EXTABLE
      USE /SWTCH/
IS      BSS 1
IB      BSS 1
      USE *
EXTABLE BSSZ 1
      SAI PLIST
LABL    RJ =XIEXDEC
      SRO B0
      SRO B0
      EQ NXT
PLIST   VPD 60/ONE
ONE     DATA 1
      BSSZ 1
NXT     SAI LABL
      BX6 X1
      SA6 IS
      SX6 TABLE.
      SA6 IB
      RJ =XSWITCHS
      EQ EXTABLE
TABLE.  BSSZ 1
      JP B3+TAB
TAB     BSS 0
      RJ =XFILATT
      EQ TABLE.
      RJ =XEXEC
      EQ TABLE.
      RJ =XPRINIT
      EQ TABLE.
      RJ =XDEINIT
      EQ TABLE.
      RJ =XPPINIT
      EQ TABLE.
      END

      FUNCTION IEXDEC(KEY)
*CALL DATBAS
*CALL ATIRIB
      IEXDEC = 0
      IF(KEY.EQ.1)GO TO 1
      IF(KEY.EQ.2)GO TO 2
      GO TO 100
C
C      IS THERE AN ACTIVE DATA BASE
C
C      1 IF(FILES(IUNITD+1).NE.0.) 100,101
C
C      IS THERE AN ACTIVE DATA BASE **AND**
C      IS STRESS IN DATA BASE FOR POSTPROCESSING
C
C      2 IF(FILES(IUNITD+1).EQ.0.) GO TO 101
      L=LENATT(3)
      IF(L.EQ.0)GO TO 101
      DO 20 I=1,L
      N=ICATT(I,3)
      N=HAMPSH(N,3)
      N=SHIFT(N,-18).AND. .NOT. MASK(24)
      IF(N.EQ. 6R N STR)GO TO 100
20  CONTINUE
      GO TO 101
100  IEXDEC = 1
101  RETURN
      END

```

FIGURE 2.4.2 - SAMPLE MENU GENERATOR OUTPUT (1)

```

1 (TEXT EDITING PROCESSOR)
  2 (PROMPTING MESSAGE)-----
  2 PREPROCESSORS
    3 (PROMPTING MESSAGE)-----
    3 GPRIME
      4 (PROMPTING MESSAGE)-----
      4 GPRIME-NEW
      4 GPRIME-RESTART
      3 OTHER PREPROCESSORS
        4 RETURN
        3 RETURN
        2 DISPLAY AND EDIT
        2 POST PROCESSORS
        2 RETURN
          CALLS FILATT      GOES TO LEVEL 2 FROM A = 2
          CHOOSE MAJOR MODULE.
          CHOOSE PREPROCESSOR
            GOES TO LEVEL 3 FROM A = 2
            GOES TO LEVEL 4 FROM A = 2
            NEW OR RESTART.
            CALLS EXEC      GOES TO LEVEL 3 FROM A = -1
            CALLS EXEC      GOES TO LEVEL 3 FROM A = -1
            CALLS PRINT     GOES TO LEVEL 4 FROM A = 2
            GOES TO LEVEL 3 FROM A = -1
            GOES TO LEVEL 2 FROM A = -1
            CALLS DEINIT    GOES TO LEVEL 3 FROM A = 2
            CALLS PPINIT    GOES TO LEVEL 3 FROM A = 2
            GOES TO LEVEL 1 FROM A = -1
                                OPTION = 1
                                OPTION = 2

```

FIGURE 2.4.3 - SAMPLE MENU GENERATOR OUTPUT (2)

15, N = "ENTER NUMBERS", S = (routine to put up numbers via PUTEDT),
A = 2, T = 5
16, L = -1, P = "ENTER NUMBERS, THEN ETX.",
S = (process type-in), A = -1,
16, N = "RETURN", A = -1, T = 6,

FIGURE 2.4.4. TRIPLET FOR TEXT EDITING

14, N = "CHOOSE BY LIGHTPEN", S = (setup ICAT and MEAN),
A = 2, T = 5,
15, L = -1, P = "CHOOSE SOMETHING, THEN RETURN.",
15, N = "RETURN", S = (routine to process each pick),
A = 1, D = -1,

FIGURE 2.4.5. MORE THAN BUTTON PICKING

2.4.3 Menu Data Base

The menu data base is in the form of a general tree. Through values provided in the tree, a subroutine is called to process the pick and take further action in the tree. The actual format and internal bead format are given in this section.

The data structure imposed on the tree requires the use of three pointers: father (pointing to the next higher level in the tree), son (pointing to the next level lower in the tree), and link (pointing to the brothers of this bead on the same level). Each bead can have only one reference to each of these pointers. A schematic diagram of the structure is given in Figure 2.4.6.

The format for the actual beads is given in Figure 2.4.7. The functions performed by each field is quite straightforward. The links are held as bead addressess in fields FTR, SON, and LNK. The traverse flag (TFL) is unused in the current versions of both the generator and driver. It has a potential use as a historical marker if such a feature is added to the driver.

Other fields are used to indicate construction of the displayed menu and the action to be taken after the user picks a menu item. There are two types of displayed items. First, each menu page can be introduced by a prompting message (PRM). The prompting message is flagged by PSX = PSY = 0. Otherwise, PSX, PSY is the location of the pick message (PCM) in raster units on the screen. The buttons on the screen can have ID blocks assigned to them for INTERTEK which are only used for special action picks. The blocks are assigned automatically by the generator for a special action pick type. A button pick is the only other pick type needed by the system. The button can be skipped if the option value (OPT) triggers the proper test in the situation decision function.

The actions taken in the program and tree after a pick has occurred is controlled by the rest of the bead. These actions are the key to the control of user interaction with the menu. After a button pick is received in the driver, a user-specified subroutine is called (RUT). Access to the routine is through the TBL field, which is an index into the automatically generated COMPASS subroutine. The user need only supply the

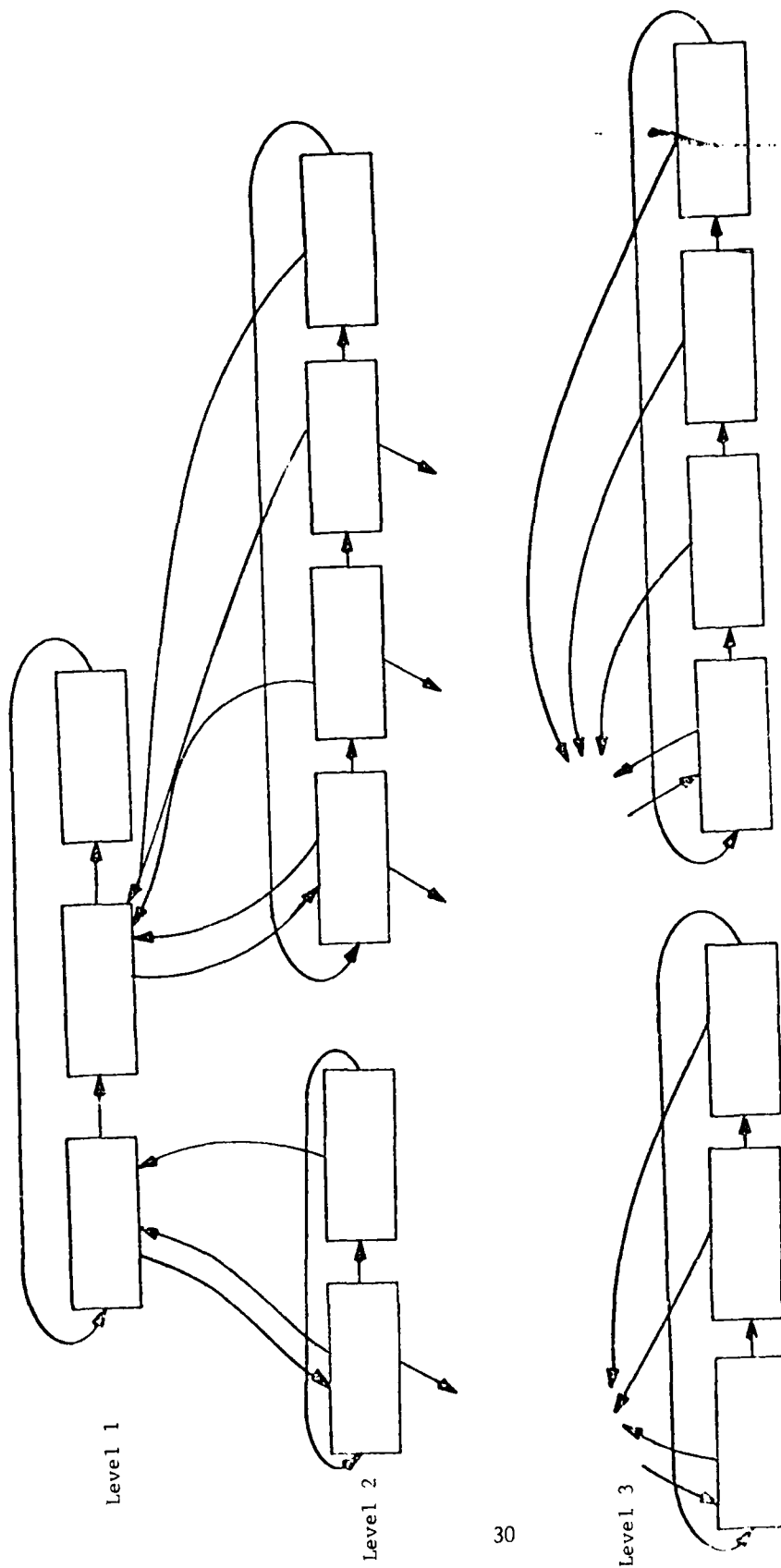


FIGURE 2.4.6 - TREE SCHEMATIC

Word	59	53	35	29	17	5	0
1	Bead Address of Father (FTR)				Bead Address of Son (SON)		
2	Traverse Flag (TFL)				Bead Address of Circular Link (LCK)		
3	Subroutine to be Called (RUT)				Address of sub-routine (TBL)		End of (SE)
4	X Position on Screen (PSX)				Y Position on Screen (PSY)		
5	Action in Tree After Return From RUT (ACT)				Bead Address of Definition (DEF)		
6	Unused	# Characters in Pick Message (PNC)	# of Words in Pick Message (PNW)		Start in Bead of Pick Message (PS)		
7	Unused	# Characters in Prompting Message (PRC)	# of Words in Prompting Message (PRN)		Start in Bead of Prompting Message (PRS)		
8	Unused	Pick Message Pick Type (PCT)	# of Words in ID Block for Pick Msg (PCN)		Start in Bead of ID Words (PCS)		
9	Unused	# of User Supplied Default Values (DVN)	Unused		Start in Bead of Default Values (DVS)		
10	Option Value (OPT)						
PS=11	Button Pick Message (PM)						
PS+PNW							
PRS	Prompting Message (PRM)						
PRS+PRN							
PCS	ID Info for Button Pick (PC)						
PCS+PCN							
DVS	User Assignable Default Words (DV)						
DVS+DVN							

FIGURE 2.4.7 - MENU DATA BASE BEAD FORMAT

name and the generator takes care of the subroutine table creation. After the processing subroutine is called by the driver, the next action in the tree is taken. The ACT field indicates which level in the tree contains the next set of buttons seen by the user. Additional values may be specified in the user-defined values (DV).

The menus operate from a random disk file and require very little information in core to keep them running smoothly. Only one bead on the level being displayed is maintained in core. Both the non-GLOBAL and GLOBAL menu require this bead to be in array ISTART in COMDECK MENBLK. ISTART(1) contains the non-GLOBAL head and ISTART(2) the GLOBAL head. (By convention, all non-GLOBAL menus are called MENU 1 and the GLOBAL menu MENU 2). The generator places the address of the first bead in level 1 of any menu in the father (FTR) field of the first bead of the file (10000B with unit information attached). Thus, ISTART(1)=IGFTR(2000010000B) will get the first bead in initializing the driver.

The driver requires that all menu bead addresses on the screen be saved in array IBEADS in common block MENBLK. The index into the array is provided by the ITASKC parameter in the GITEM call for each button name. When switching from NON-GLOBAL to GLOBAL mode, IBEADS must be saved and restored before returning back to non-GLOBAL mode.

The driver passes information about picks to the called subroutine using the COMDECK IDENTs. All information about the pick is passed to the subroutine in the IEVENT, ID, and INF arrays. By convention, a return of ID(1)=0 (in addition to the standard INTERTEK return of IEVENT=-1) means that no pick has been processed.

2.4.4 Menu Generation Software

The menu generation software is on program library file MENUGENPL. The absolute for this system is file MENUGENABS which is created in procedure MENUGENABS. It is invoked by procedure MENUGEN whenever a new command tree is to be created. The subroutines on MENUGENPL are listed in Table 2.4.1.

TABLE 2.4.1 MENU GENERATION SOFTWARE

TMENU	Main program
GENTREE	Generate menu data base and source code for menu driver and decision function
GETSTR	Input free format input stream
CRKSTR	Crack input stream into proper fields
FINSPC	Look for special action names such as "RETURN"
IFNDH	Generate new special action definition
IGFTR	Get menu data base bead fields using NSRDC Data Handler
IGPCKM	Get other menu data base fields
ISTUFF	Look for menu level number on input stream; echo input stream back to OUTPUT if requested
MAKALF	Encode an alphanumeric string
MAKINT	Convert input display code number to an integer
PROUT	Generate jump table of routines called from this menu
SETFTR	Put menu data base fields using NSRDC Data Handler
SETLN	Set special action definition bead
SETVAL	Set special action bead fields
DISTREE	Print a formatted listing of the tree

2.4.5 Processing the Command Tree in STAGING Driver

Because of the work done by the command tree, the driver is not much more than a sophisticated computed GO TO processor. The programmer (although he does not have to explicitly worry about it) has set up all the linkage needed to proceed smoothly through the tree. The driver merely puts the buttons on the screen, then waits for a button pick event. Upon receipt of the pick, the driver retrieves the subroutine associated with the button and processes any intermediate picks (single, string, parameter, or text editing type events). Each intermediate pick is processed by the subroutine specified in the button pick name. After intermediate picks are processed, the routine is called for clean up operations. If the routine detects an error, the next action in the tree can be overridden. The tree action is specified during the generation phase and lets the programmer specify where he wants the user to go next. Actions are basically go up, go down, and stay on the same level.

The driver also takes care of initialization and abnormal termination for STAGING. The steps in initialization are:

- (1) Initialize the common blocks (GLOINT)
- (2) Initialize the recovery package (MARK)
- (3) Initialize graphics (GRFNIT)
- (4) Initialize the Executive and Global trees (MENNIT and STARTM)

DRIVER then loops through processing all picks. There is never any need for a subroutine to include a pick processing call (GIBUTN, GIPARM, GISTRN, GISNGL) outside of DRIVER.

If an abnormal termination is detected, DRIVER and the recovery package are responsible for either stopping the application or continuing in a reasonable way. Stopping the application simply returns to INTERCOM without flushing the user's data base. Restarting reinitializes graphics (GRFNIT), the GLOBAL menus (MENITG), and deactivates all active beads in the data base (ERASE).

Routines that are directly related to DRIVER include:

- IPPRMT--display prompting message on this level
- IGFTR with entry points

IGACTION, IGDEF
 IGDVN, IGDVS, IGLNK,
 IGPRS, IGPRW, IGPS,
 IPSX, IGPSY, IGRUT
 IGSFT, IGSON, IGTBL,
 IGTFL, IGOPT
 IGPCCKM--retrieve pick message
 with entries
 IGPRMM--prompting message
 IGDFV--default values
 IGIDM--ID values (special actions)
 SWITCH--call subroutine specified in command tree
 with entry
 SWITCHS--save address of start of subroutine table and
 decision function (provided by menu generator)
 ISWTCHD--call the situation decision function
 GRFNIT--initialize graphics for INTERTEK
 MENNIT--initialize the Global and Executive menus by attaching
 the proper files, initiating the subroutines, and
 drawing the first display
 With entry
 EXINIT--reinitialize menu DAE and GLOBAL button after a
 recovery has been made
 GLOINT--initialize common block variables and INTERTEK buffer
 EXINIT--initialize the executive menu upon return from a
 submodule
 STARTM--start up the menu on the specified local file name
 (once a command tree is initiated by the Executive
 module, it functions as a separate entity until a return
 to the Executive is requested by a call to EXINIT).
 INDATA--read text input and format it for cracking algorithms
 IREAD --read a text string into NVALS in EDITT common block
 IPICK --process button pick
 PAGER --wait for user to pick ERASE or REDRAW
 ERASEM--erase and redraw

ERASER--erase screen and redraw only minimal DAE's
with entry
ERASEN--erase screen, no redraw

2.4.6 Textual Input

Textual input to STAGING is needed for user modification of fields displayed in the menu area. A general mechanism has been defined for preparing these areas that simplifies coding.

The number of different entries for text editing throughout STAGING led to the development of a generalized subroutine for displaying the items to be edited.

PUTEDT: Display a list in the type-in area for text editing. The programs can supply up to ten values in one call. The length supplied can be positive or negative. A positive value indicates that all values in the array should be displayed even if there was no old value. A negative value means that blank old values are ignored. This mode is used primarily when erroneous values are being redisplayed to let the user correct only those values. As well as the length of the type-in, the program must supply a Hollerith descriptor for the field in array NAMES in common block EDITT. An old value can be supplied in array LVALS. These values must also be Hollerith. PUTEDT takes care of the formatting and display. The user should set IST in EDITT to the beginning of the type-in for each block and IEND in EDITT to the maximum number in the list. PUTEDT causes the type-in box to be displayed.

As with the name display, the type-in area is limited to ten items. The same general tactics are used to put up the next sets of values: an entry point in the routine that put up the first set puts up subsequent values.

Whenever text is input by the STAGING user, the processing subroutine must check the validity of the entry as well as performing the

proper action. Each type-in has different requirements for validity, so the actual checking has to be spread into each processing routine. Utilities are provided to help decode the values the user types into NVALs. For each new type-in, NVALs will be non-blank. The usual construction of the processing routine is:

1. Check to see if all numeric input is correct.
If so, go to step 3.
2. Check to see if any values are special case alpha-numeric inputs. If no more errors, go to step 3, else, display an error message, stay on the same level and call PUTEDT again to redisplay only the bad values. The loop is continued until the user 'RETURNS' or succeeds in the type-in.
3. Check to see if more values need to be entered. If so, call the put up entry point and return control to DRIVER. Note that the entry point always bumps IST in EDITT.
4. Compact the type-in list so all values are contiguous.
5. Do processing for all collected values.

The practical limit for a single set to be typed in is 30 (the length of SVALL and SVALH in EDITT). Each correctly decoded value is stored in these arrays biased by IST so no more than 10 values can be typed before processing begins. The following routines are used for processing type-ins:

ICHKED: Check to see if all numeric input is correct. Two modes are allowed: single value or range type-in. The range type-in is two numbers separated by the letter 't' or the word 'to'. If an error occurs, the new value is placed in LVALS and the function returns a non-zero value. PUTEDT will then redisplay the erroneous values with no further programming. The numeric values are always floating point and stored in SVALL (low) and SVALH (high) in EDITT.

COMPAC: When all typing is completed, the value can be processed more conveniently when no searching needs to be done. COMPAC, therefore, squeezes the type-ins together to allow a loop index of 1 to IST in EDITT. (Always check

if IST = 0 for no type-in). NAMES (1-IST) in EDITT contains the actual location in the list at which the type in was made.

Review of the following routines will aid in understanding the processing of text input. For routines that can handle more than 10 values, look at routines PUTATT and ASERCH. For a routine that handles special alphanumeric values, review SVIEWP.

2.5 Staging Model Data Base

2.5.1 Introduction

The STAGING model data base is split into four geometry levels, each of which contains a description of part of a model. Level 4 (nodes) describes points in space in terms of (X,Y,Z) coordinates. Level 3 collects nodes into elements, level 2 collects elements into substructures, and level 1 collects substructures into structures. Each bead in a level is linked to the bead in front and in back of it. It also contains pointers to its direct ancestor and descendent beads, thus providing a totally cross-referenced directed graph structure. User values are stored as attributes in a variable length array assigned to each bead.

An additional level (#5) has been added to the data base to allow storage of non-geometric data which can be anything from card images to large arrays of numeric values. Each array can be given a unique forty-character name. It is convenient to consider the tables as FORTRAN arrays: they are created by a conversion routine, can be of 1, 2, or 3 dimensions, and cannot increase dynamically in size.

Utilities are provided at different levels to store and retrieve information in the data base. The data base handler subroutines provide diverse functions for manipulation of the data base. Searching for structures, substructures, and table names is done linearly. However, in order to allow faster searches, nodes and elements are maintained in numeric order. Description of the search procedures for nodes and elements is given in Section 2.5.4.1.

2.5.2 STAGING Model Data Base Format

The overall structure of a STAGING model data base is illustrated in Figure 2.5.1. Each level in the structure is linked internally within that level by two-way circular pointers. The left link points to the element (or node) with the next smallest numeric identifier; the right link points to the next highest. The structures and substructures are

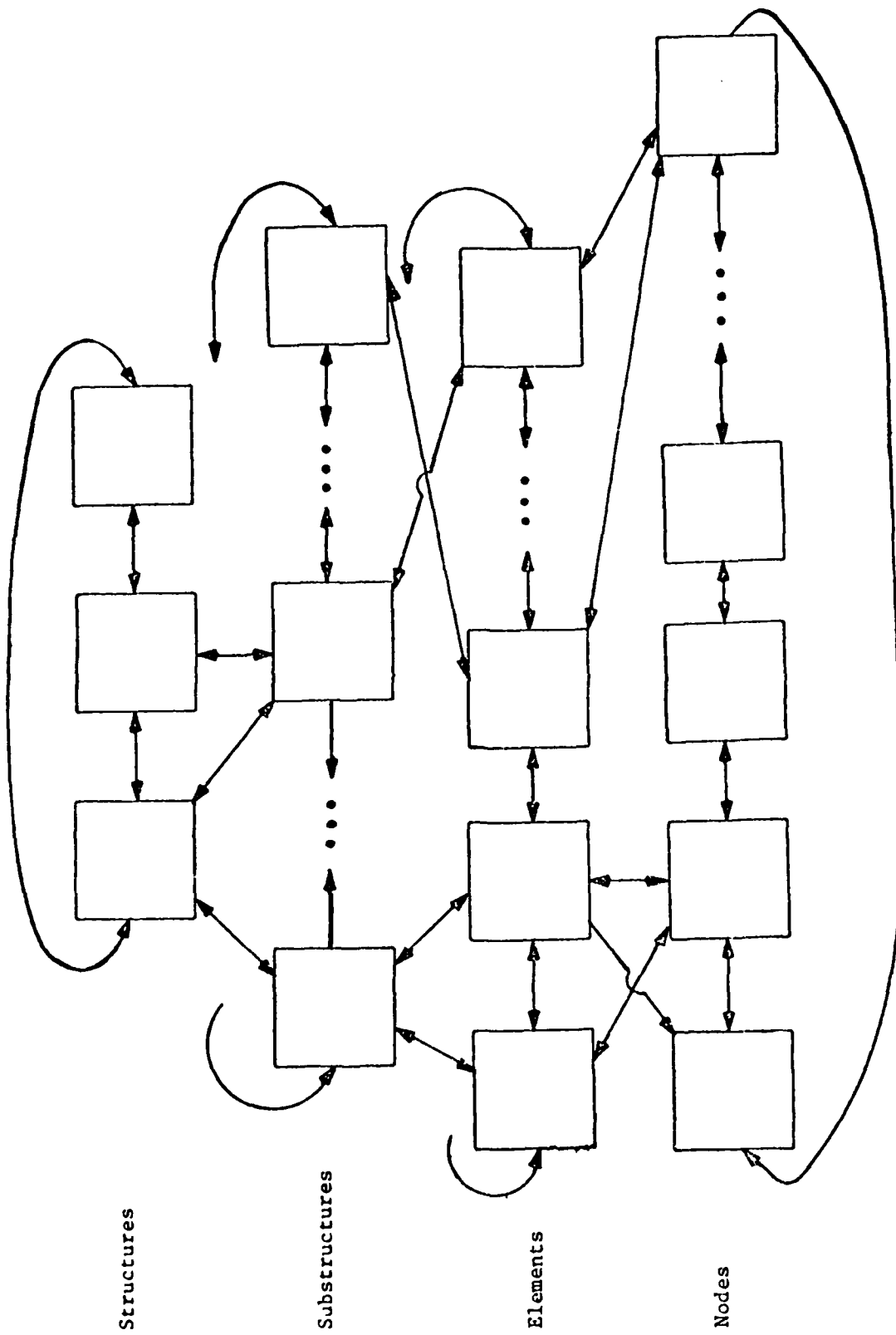


FIGURE 2.5.1 - OVERALL STRUCTURE OF BEADS

unordered but similarly linked. Another list is formed on each level that joins each active bead on that level. This list is unsorted and is built as new beads on a level are activated. The levels are joined to the next higher and lower level. The data base is completely cross-referenced between levels. If element 2 contains node 6, then node 6 has an up pointer to element 2. This helps when a change is made to a node and the change effects how an element is drawn. No searching is necessary to find the ancestors.

The goal of all this linkage is to minimize search time. The only time an entire list needs to be searched for a name is for the relatively short structure and substructure lists. By keeping a partial table in-core for elements and nodes, a number can be found quickly. Furthermore, only the address of the starting and ending beads need to be found if a range of numbers has been entered (for example, if 10 to 50 is entered all numbers within that range are known immediately because the list is sorted). A detailed description of the table is given in Section 2.5.4.1

2.5.2.2 Detailed Bead Description

The internal structure of the beads is the same for each level from 1 to 4. The general format for these beads is given in Figure 2.5.2. Figure 2.5.3 shows the format for a level 5 table bead.

The fields in the bead are interpreted as:

Word 1 TYPE (6 bits, sign extended-TYP) - The type or level number (1-4) of the bead. If TYPE is negative, the bead has had some value changed during editing. Thus, when just concerned with the type, one must take the absolute value before using the value.

DISPLAY ATTRIBUTE (6 bits, not sign extended - DAC). The attribute being displayed if the bead is active. If =0, no attribute is displayed; if LENATT(TYPE), the name or number is displayed. If DAC is between 0 and LENATT(TYPE), the value is the location in the attribute list of value shown.

LENGTH (18 bits, not sign extended-LEN). The total number of words in the bead.

	59	53	47	29	17	0
1	TYPE	DISPLAY ATTRIBUTE			LENGTH	
2		ELEMENT OR NODE NUMBER				
3		LEFT DATA BASE POINTER		RIGHT DATA BASE POINTER		
4		IDDAD #1 (DAD)		START OF ATTRIBUTE LIST (STA)		
5		IDDAD #2 (DAD)		NUMBER OF DOWN POINTERS (DNN)		
6		NUMBER OF UP POINTERS (UPN)		START OF UP POINTERS (UPS)		
LPROLG = 7		LEFT ACTIVE POINTER		RIGHT ACTIVE POINTER		
LPROLG + 1		DOWN POINTER #1		DOWN POINTER #2		
				.		
(DNN-1)/2+1+LPROLG		DOWN POINTER #DNN-1		DOWN POINTER #DNN		
UPS		UP POINTER #1		UP POINTER #2		
				.		
(UPN-1)/2+UPS		UP POINTER #UPN				
STA		NO MAN'S LAND				
STA+JDEF(TYPE)-1		ATTRIBUTE VALUE #1				
STA+JDEF(TYPE)				.		
				.		
STA+JDEF(TYPE) +LENATT(TYPE)-1		ATTRIBUTE VALUE #LENATT(TYPE)				
LENGTH		FREE SPACE OF VARIABLE LENGTH				

FIGURE 2.5.2 - FORMAT OF DATA BASE BEADS

Word	59	17	0	
1	TYPE*		LENGTH*	
2	unused			
3	Left Data Base Pointer*	Right Data Base Pointer*		
4	unused	Start of Data List* (STA)		
5	unused			
6	unused			
LPROLG=7	Left Active Pointer*	Right Active Pointer*		
8	Name of Table (NAM)* [4 Words]			} No- Man's Land
9				
10				
11				
12	Number of Dimensions (NDM)			
13	Row Dimension (ROW)			
14	Column Dimension (COL)			
15	Depth Dimension (DEP)			
16	User Supplied Format (TFM) [3 Words]			
17				
18	Type of Format (FTY)			
LPROLG+JDEF(5) 19				
STA 20	User Supplied Data . . .			
LENGTH				

No-
Man's
Land

*Function for setting and getting done by same routine as Levels 1 through 4.

NDM: the dimensionality of the supplied array.

ROW: number of rows in table. If NOM = 1, ROW is the length of the array.

COL: number of columns in table. If NOM = 1, COL is zero.

DEP: length of depth in table. If NOM = 3, DEP is zero.

TFM: variable format in characters that can be up to 30 characters long.

FTY: if TFM is of A format, FTY = 0, else 1.

FIGURE 2.5.3 - TABLE BEAD FORMAT

- Word 2 If TYPE = 3 or 4, the element or node number in integer format. If TYPE = 1 or 2, the name is stored in 'NO MAN'S LAND' (word STA through STA + 3).
- Word 3 LEFT DATA BASE POINTER (30 bits, not sign extended - LFT). If TYPE = 3 or 4, bead address of the element or node with first number less than number in word 2. If TYPE = 1 or 2, bead address of previously created structure or substructure. If bead is the head of the list, LFT points to the last bead in the list. RIGHT DATA BASE POINTER (30 bits, not sign extended - RIT). If TYPE = 3 or 4, bead address of the element or node with first number greater than number in word 2. If TYPE = 1 or 2, bead address of bead created immediately after this structure or substructure. If this bead is at the end of the level, RIT points to the head of the list.
- Word 4 IDDAAD #1 (30 bits, sign extended - DAD). IDDAAD #1 is address of display items for undeformed plot. If value is negative, the plot is shrunk 80% about its center. If DAD = 0, the item is not drawn. The various drawing routines realize that an item is already drawn unless DAD=0. A temporary set of DAD to 1 is made to tell the drawing software that GUSETP has been called. When an item is generated, DAD is not the proper value for the generation. An active-no-draw mode is used in 2D and contour plotting. In this case, DAD is set to 1 to inhibit drawing. START OF ATTRIBUTE LIST (3 bits, integer value - STA). Word in bead in which first attribute starts.
- Word 5 IDDAAD #2 (30 bits, sign extended - DAD). Display item address of deformed picture of entity. NUMBER OF DOWN POINTERS (30 bits, integer value - DNN). Number of down points in this bead. This list always starts in word LPROLG + 1. Nodes have no down pointers.
- Word 6 NUMBER OF UP POINTERS (30 bits, not sign extended - UPN). Number of up pointers in this bead. Structures have no up pointers. STARTOF UP POINTERS (30 bits, not sign extended - UPS). Start in bead of up pointers. Keeping this value allows gaps in the bead between the down pointers and up pointers.

Word 7 LEFT ACTIVE POINTER (30 bits, not sign extended - DLF). If a bead is activated DLF contains the previously activated bead address. If this bead is the first active pointer bead on this level, DLF points to the last active bead.

RIGHT ACTIVE POINTER (30 bits, not sign extended - DRT). If a bead is activated, DRT contains the bead address of the next active bead. If this bead is the last active bead, DRT points to the head of the active list. To determine if a bead is active, the right pointer (DRT) needs to be non-zero. The system takes care to reset the right pointer to zero when a checkpoint or stop is taken. That insures that the file is usable at a later date. When a bead is activated the DAD and DAC parameters are set properly.

The above 7 words are always present in each bead at each geometric level.

DOWN POINTERS

The pointers are bead addresses packed two per word. The list is of variable length and must be compact (i.e. no holes in the pointer list). Therefore, when a down pointer is deleted, all of its successors are moved up a slot and the number of down pointers decreased. The deletion process only moves down pointers because the start of the up pointers (UPS) and attribute list (STA) is independent of the number of down pointers. Holes between the down pointer list, the up pointer list, and the attribute list are possible because of this.

LIST OF UP POINTERS

The up pointer list works exactly the same way as the down pointer list. It is packed, two-bead addresses per word, and of variable length.

NO MAN'S LAND

This area is a region that serves as a catch-all for any other types of information needed in any of the levels. Words are stored and retrieved through array fetches from the Data Manager package. Each bead of a particular case reserves JDEF(TYPE) words if that area of the system needs

additional information. The current implementation uses JDEF lengths of four in a structure, ten in a substructure and zero for both elements and nodes. In the structure, words 1 through 4 are taken by the forty-character name. The substructure also keeps the name in words 1 through 4. Words 5 through 7 are the X, Y, and Z minimum coordinate values for the entire structure or substructure while words 8 through 10 contain the X, Y, and Z maximum coordinate values.

The JDEF region of no-man's land can be used for other information as the need arises.

ATTRIBUTE LIST

The actual floating point values of the attributes. The order is pointed to in the COMMON block ATTRIB which will be described later. Each location points to a new value in the list of possible attributes. The order and meaning of the attributes is defined by the user in the conversion routine CNINIT. Additional attributes can be added through later conversion routines in a manner transparent to the programmer. He must merely assign a value to an attribute not defined in the initial data base, and that new attribute will be defined for the data base.

2.5.3 Change List

One of the features of the general data base is the ability to back up to a previously saved data base if editing does not prove satisfactory. The change list resides on the same data handler file as the model data base.

The concept behind the change list is to save only the current version of each bead in the data base itself. Thus, when, a change is made to a field, the change is reflected in the bead itself. If this is

the first change made to a bead, an unchanged copy of that bead is placed on the change list. The bead format for the change list is given in Figure 2.5.4.

Entries are made to the change list only for those fields that relate directly to the data base. These fields include LENGTH, NUMBER or NAME, LEFT or RIGHT DATA BASE POINTERS, START OF ATTRIBUTE or UP POINTERS, NUMBER OF UP or DOWN POINTERS, and UP or DOWN POINTER, or any ATTRIBUTE VALUE. The display related fields of DISPLAY ATTRIBUTE, IDAD, LEFT or RIGHT ACTIVE POINTER, and MINS and MAXES (substructure beads only) do not cause entries to the change file. To inhibit searches for changed beads, the system flags a changed bead by setting TYPE to -TYPE.

The change list does not have to be used unless the ability to back up is desired. For example, the conversion routines inhibit use of the change list by setting the header of the list to zero. When the change list is active, as it is in Display and Edit, the header of the change list is a bead as formatted in Figure 2.5.5. The user tells the system that a change file is needed by setting the second parameter in the data base initiation routine (DBINIT) to zero. The conversion routine initiator (CNINIT) automatically sets the change file inactive mode. If the change field is inactive, beads are released whenever possible.

The headers and number of beads are kept to insure that deletions or additions to the data base are properly eliminated when backing up to an old data base.

The change file works for all cases, even when a new bead of greater length is needed. Because all references to the new bead are also updated automatically and the old bead is unreleased, the change file need only rewrite the old beads again. The linkage is thus automatically restored to its original state. When the change file is 'released' and the user saves his edited data base, the list is destroyed by making the last bead in the list the same as the first bead. The change beads themselves are not released to give a future capability of backing up more than one step.

The programmer need not worry about using the change file as long as the model data base change routines are used. These routines automatically dump beads to the change file when necessary.

Word	
1	Length of Bead
2	Bead Address of Next Bead in List*
3	Bead Address of Original Bead
4	exact copy of Bead in address in Word 3 <u>Before</u> a change is Made
Length	

* If this is the last bead in the list, next bead points to the head of the list.

FIGURE 2.5.4 - CHANGE BEAD FORMAT

Word	
1	Unused
2	Bead Address of Next Bead in List
3	Unused
4	Head of Level 1
5	Head of Level 2
6	Head of Level 3
7	Head of Level 4
8	Number of Beads in Level 1
9	Number of Beads in Level 2
10	Number of Beads in Level 3
11	Number of Beads in Level 4

FIGURE 2.5.5 - CHANGE FILE HEADER BEAD

2.5.4 In-Core Tables and Arrays

All in-core arrays are in common blocks that are in UPDATE COMDECK's. These hold the values needed to use the data bases effectively.

2.5.4.1 Searching Tables

The conversion routines used in the construction phase maintain several tables. The user may supply nodes and elements in any order. The conversion routines keep a table (ELNOD in CNINT) that contains mapped elements and node beads by number and associated bead address. The table fills from the front for elements and from the rear for nodes as depicted in Figure 2.5.6. When the table overflows, a copy is written to the conversion routine scratch file (TAPE77). CNTERM sorts each full table and links them properly. Obviously, the conversion routines operate faster if the nodes and elements enter in sorted order.

One of the features of the model data base is fast search times for node and element numbers. Because the circular list is kept in order, a binary search can be done based on a partial table of entries. The strategy is to minimize disk accesses. The number of accesses goes up slowly as the number of elements and nodes increases by sophisticated use of the table in ILKUP and INLKUP in DATBAS. The actual table format is given in Figure 2.5.7.

The number of entries in ILKUP depends on the size of the data base. The number of accesses depends on how close the entry is to the entry in the ILKUP table. If the number is not in the table, a guess is made as to which entry the search number is closest to, thus further decreasing search time. For example suppose there are 500 elements and 500 nodes in the data base, both numbered from 1 to 500 (the numbering scheme will not effect the algorithm at all). Suppose we are looking for the bead address corresponding to node 87. The ILKUP table would have entries for the tenth bead in each list because ILKUP of dimension 400 is split into four equal pieces. Node and element numbers 1, 11, 21, ..., 491 would all have references directly in the table. A binary search for 87 indicates that 81 is its predecessor in the table. The algorithm then

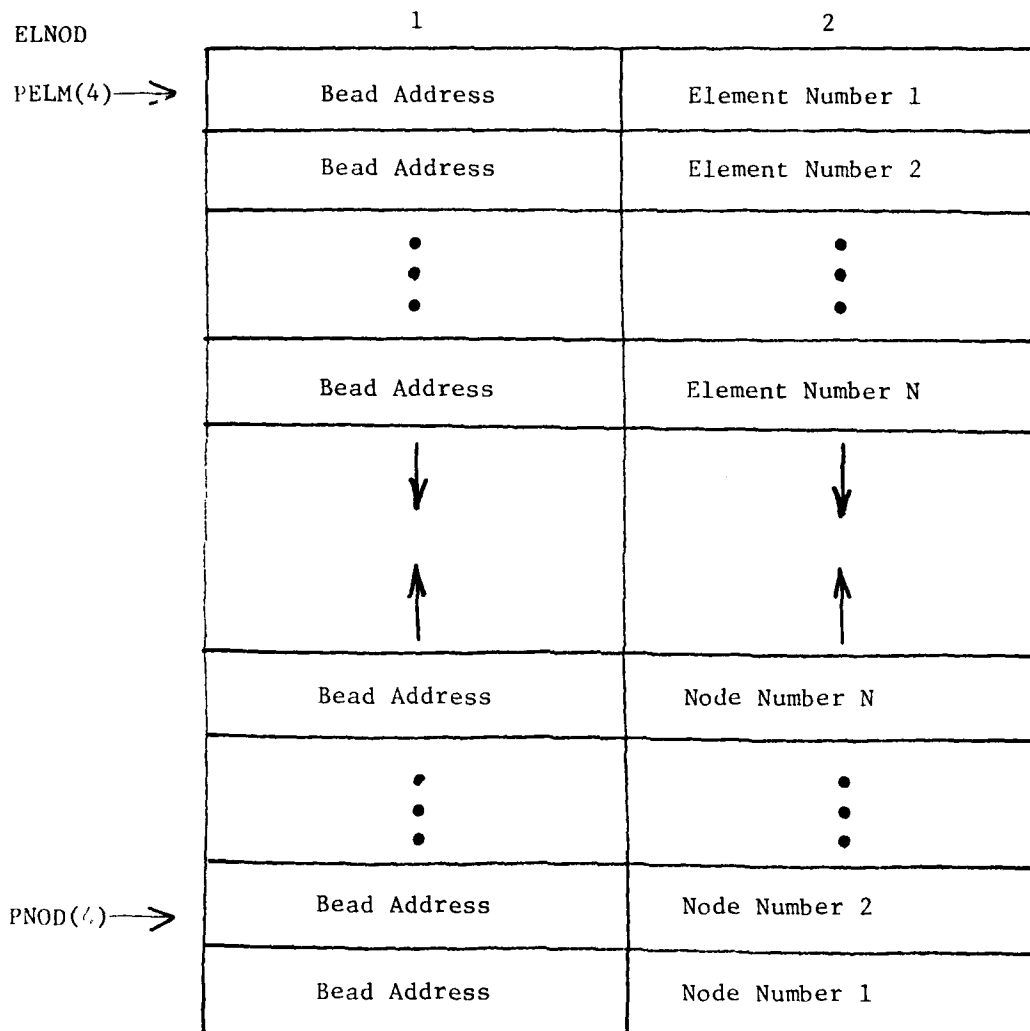


FIGURE 2.5.6. FILLING ELNOD

INLKUP(1)	ILKUP Element Number A
INLKUP(1)+INLKUP(3)-1 INLKUP(1)+INLKUP(3)	Element Number 2 Bead Address Corresponding to Element A
INLKUP(1)+2*INLKUP(3)-1 INLKUP(2)	Bead Address Corresponding to Element Z Node Number A
INLKUP(2)+INLKUP(4)-1 INLKUP(2)+INLKUP(4)	Node Number Z Bead Address Corresponding to Node A
INLKUP(2)+2*INLKUP(4)-1	Bead Address Corresponding to Node Z

FIGURE 2.5.7 - ILKUP ARRAY WITH INDEX INLKUP

looks at the successor in the table, in this case 91. A direct comparison determines that 87 is numerically closer to 91 than 81. A total of three beads would be fetched (90,89,88) before the hit was made and the fourth was discovered to be correct. A maximum of five beads need to be checked to get any one bead for this data base, with average search time for a bead not in the table 2-1/2 beads checked. The algorithm is general: all one has to do to increase speed for huge problems is increase the size of ILKUP and set the new length in DATBAS variable LELKUP. It should be noted that 200 words seems quite adequate because not that many searches are made for name in Display and Edit. The hardwiring of bead addresses as links greatly speeds up the processing time and seems most adequate. The conversion routine CNTERM uses the node and element names to generate up and down pointers. This process has been speeded up by sorting the down pointers before searching, thus eliminating searches for duplicate names. The scheme uses core effectively (by not using much) and eliminates shuffling hash tables in and out of core.

The two links (level links and active links) have a record of the first bead (the header) and the number of beads in each list. These are kept in array of length four, with the subscript referring to the data base level (1=structure, 2=structure, 3=element, 4=node). In common block DATBAS, array IHEAD refers to the level header and NUMBDS the total number of beads on the level. In common block ASTSTR, array IHAC is the head and NAC the number of active beads on each level.

The head and tail of the change file are kept as IHCH and ILCH (respectively) in common block CHDAT.

2.5.4.2 Attributes

The attributes that can be defined for any data base are variable. A list of attributes available to the application is maintained through the variables in common block ATTRIB, ELNAC, and IATTYP.

Common block ATRRIB contains the master table of attribute names for all four levels. The names are kept in array IATNM. There are seventy-nine attributes currently available in the data base. The number of each type is kept in array MAXATT. The first attribute in the packed

(i.e. no gaps between names) IATNM array for each level is kept in array INXATT. The total scheme is depicted in Figure 2.5.8.

Each 10 character attribute name is unique up to seven characters. The last three characters are reserved for a subscript ranging from 1 to 999. In this manner, each attribute can be considered to be an array that can be accessed by a seven character mnemonic and a subscript value. (The treatment of the name implies that 0 is equivalent to 1, i.e., attribute TEMPERATUR = TEMPERA(0) = TEMPERA(1). The numbers are treated as characters internally. Blanks within the numbers are insignificant (1 1 = 11 = 11 = number 11).

The actual attribute list is order independent between data bases. Each data base defines its attribute order at initialization by routine CNINIT. The user supplies the list of which attribute names are to be assigned with which attribute value. A key is formed from each attribute name which is *60 bits long*:

59	29	0
Subscript value	pointer to name	

The pointer to the name is a pointer to the Hollerith identifier in array IATNM. The subscript value is the Nth member of the array. Two routines handle encoding and decoding of the names:

NAMFNM: Make a Hollerith string from a value in the above format

NUMFNM: make a number in the above format from a Hollerith attribute descriptor.

The actual association list is kept in array INATT. (See Figure 2.5.9). However, INATT can overflow. This condition occurs when LENATT (LEVEL) is greater than MAXATT(LEVEL). In this case, additional space is required to store the indices.

IGATT: returns Nth attribute value from array INATT. If this value is not in INATT already, IGATT fetches the proper page from memory. See Figure 2.5.10 for a description of the paging scheme. For each level, IBXATT(LEVEL) refers to the bead containing the page. Each new page adds MAXATT(LEVEL) more words to the array. IPATT(LEVEL) contains the index of the first word of the in-core page.

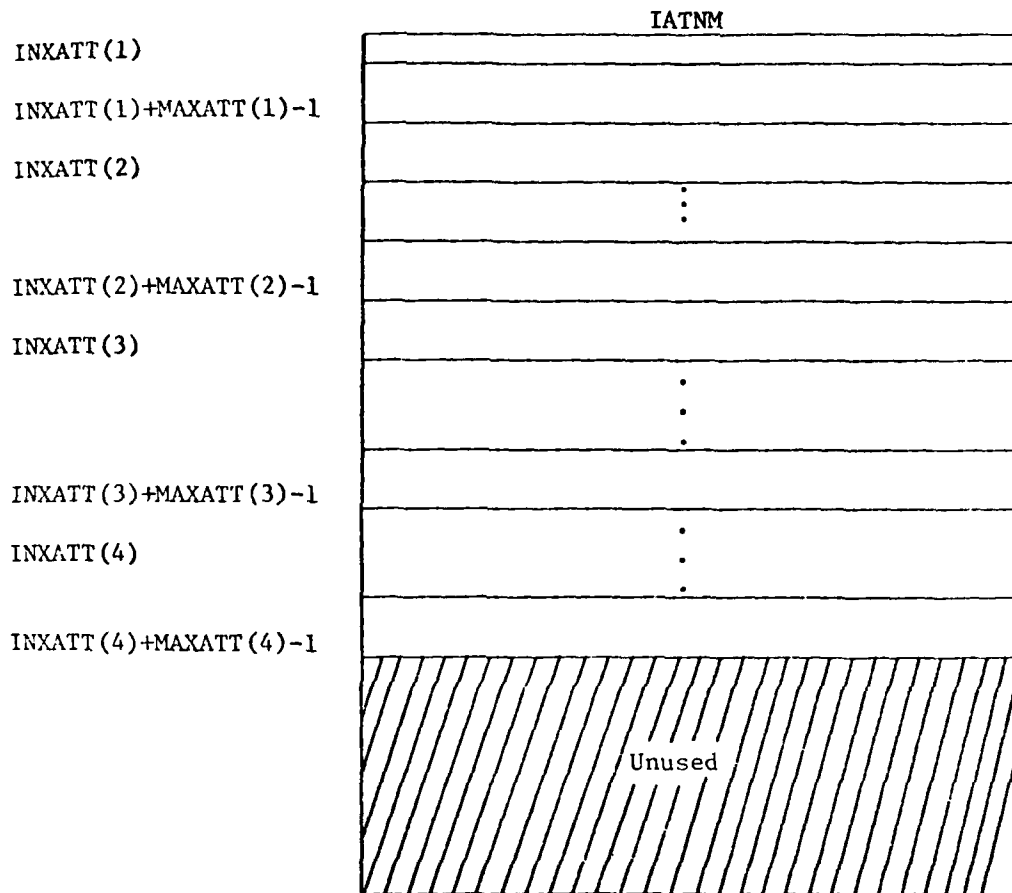


FIGURE 2.5.8 - ATTRIBUTE NAMES

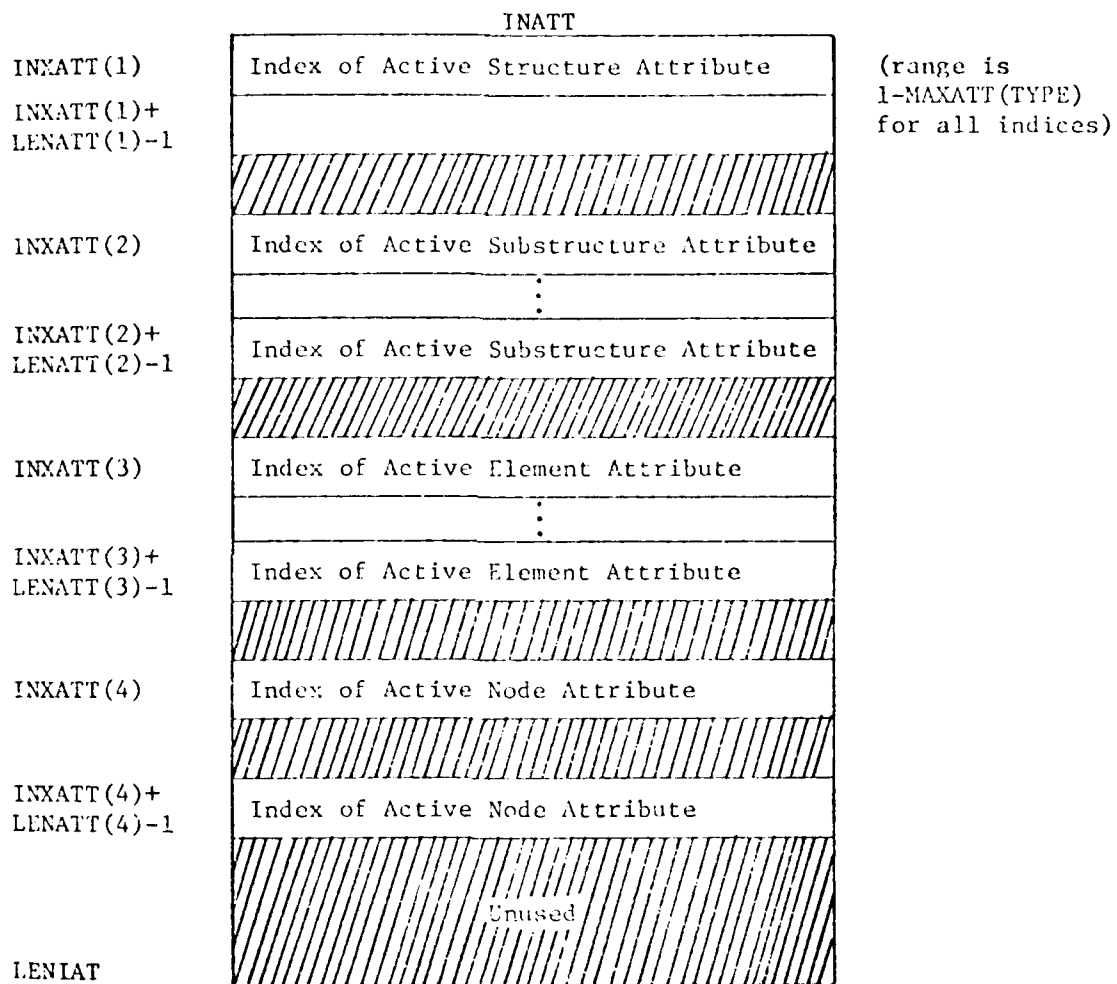
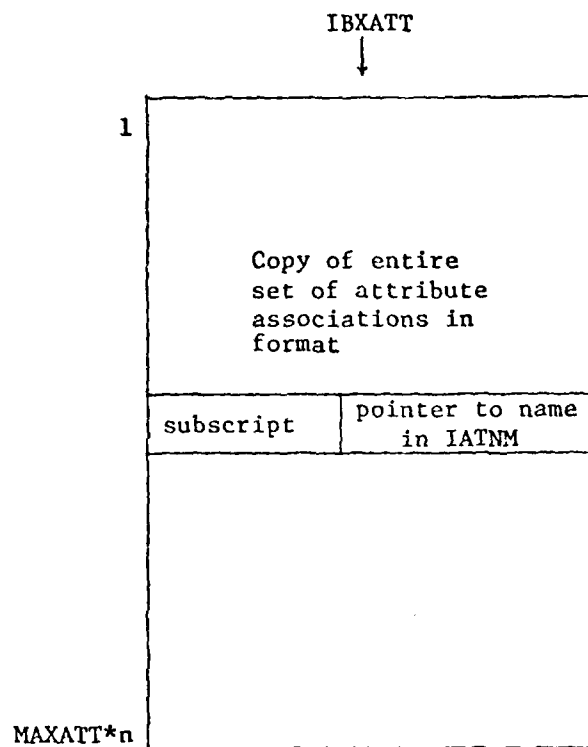


FIGURE 2.5.9 - ATTRIBUTE INDEX INTO BEA3



IPATT points to $(n - 1) * \text{MAXATT} + 1$ where n is the number of pages needed to store LENATT attributes.

FIGURE 2.5.10 - PAGING SCHEME. PAGES ARE GENERATED IF
LENATT (LEVEL) > MAXATT (LEVEL) WHEN NEW
ATTRIBUTES ARE GENERATED

The data base remembers internally which attributes are defined by using the RESTART and DBINIT routines. RESTART dumps the significant arrays to disk (especially the information about heads of lists, the LKUP array, and attributes). DBINIT restores the arrays to the same state as they were when RESTART was called.

The only attribute that must be defined is the TYPE of element. Twenty-nine different types are available in Display and Edit thus far. Array IELNAC in COMDECK ELEMEN contains a 10-character name for each element type and a key as to how the element will be drawn in the IDRWEI routine.

Display and Edit also allows display of attribute values on the model itself. To aid in the identification, the IATCOM array in COMDECK IATTYP contains an abbreviation of the full attribute name. Array IATDRW contains a flag indication how a particular attribute will be drawn (if a symbol is available). The drawing is currently restricted to single and double headed arrows.

2.5.5 Model Data Base Handler Routines

The data base can be accessed from any level desired. The low-level routines are the bread-and-butter routines that store and retrieve information from the data base. Higher level routines have been coded to make retrieval from the data base easier. They use the same low-level routines but provide short cuts to decrease coding for heavily used concepts.

2.5.5.1 Low-Level Routines

The purpose of the low-level routines is to make each field in the bead easily accessible. Each field can be set or retrieved. The set routines check to see if the bead has been changed previously. See the change file description in Section 2.5.3. All retrieving (except for arrays) return the value as the function name.

The major idea behind the plethora of names is to insulate the programmer from the bit manipulating calls of the NSRDC Data Handler. The

input that controls the format for each bead is set in data statements in GLOINT for common block CODES. In order to change the bead format, only this common block need be modified. Furthermore, this limits the number of DM- calls to allow easier conversion to another random access package on another machine.

Each set routine (prefix DB) has a counterpart get routine (prefix IDB). In the following list, the set routine is listed first.

DBCHA - DBATT: Nth attribute (DBATT returns a floating point value)
DBRITE - IDBRIT: Right pointer (next bead in list)
DBDAC - IDBDAC: Active attribute (for display purposes)
DBDAD - IDBDAD: Address of bead on screen (for display purposes)
DBDLF - IDBDLF: Left active pointer (for display purposes)
DBDNN - IDBDNN: Number of down pointers
DBDNP - IDBDNP: Nth down pointer
DBDRT - IDBDRT: Right active pointer (for display purposes)
DBLEFT - IDBLFT: Left pointer (previous bead in list)
DBLEN - IDBLEN: Length of bead in words
DBNAME - IDBNAM: Name of bead
DBSET - IDBGET: Nth word of bead
DBSTAT - IDBSTA: Start of attribute list
DBTYP - IDBYTP: Type of bead (1,2,3 or 4)
DBUPN - ID3UPN: Number of up pointers
DBUPP - IDBUPP: Nth up pointer
DBUBS - IDBUPS: Start of up pointers
DBARR - IDBARR: Array from no man's land.

The above routines access each field in the data base beads. At a slightly higher level are routines to perform more complicated tasks. These include:

DBCHNM: Change the name of an existing bead. This routine insures that the name is in the proper order for nodes and elements.
DBUPT: Set or remove the up pointer of the bead listed. The number of pointers is increased or decreased appropriately. Duplicate pointers are ignored.

DBDNT: Same as DBUPT, but for down pointers

Beads need to be added to or deleted from the data base on each level.

IDBADB: Add a bead with name in parameter 1 after the bead in parameter 2 (usually found from IDBFND--see below). If parameter 2 is a level number (1,2,3,4) and the list is empty, a new list is started with the new bead as the header. Adding a bead sets the proper left and right pointers but does not set up or down pointers or attribute values.

DBDEL: Delete the bead and all references to it including up, down, left, and right pointers. Deleting a bead does not eliminate in-core references to it.

One of the most common tasks is to find a particular name on a level. Function IDBFND provides that capability.

IDBFND: Find the bead address for level in parameter 1 corresponding to 40-character name (structures and substructures) or integer number (elements and nodes). Structures and substructures have short enough lists to search linearly. Elements and nodes are stored in sorted order and indexed by a partial array called ILKUP. A binary search is performed on ILKUP, and a linear search thereafter.

DBLKUP: The application will delete and add beads. To clean up the ILKUP array, DBLKUP will reset the array.

When adding values to a bead, overflow may occur. This can occur when setting a new value in any of the variable length lists in the bead: the up and down pointers and the attribute list. The following routine performs overflow checks:

IDBCAN: Check to see if more words can be added to the bead. If not, IDBCAN obtains more words and changes all references to the old bead, even those in in-core arrays. The new bead address is returned. The routine is called automatically by the routine concerned and is thus transparent for the most part.

The change list is established as a backup to let the user retreat to a previous version of his data base. The change list may also be incorporated into the existing data base. This allows the current data base to be the permanent data base. The following routines perform this function:

- DBCHBD: Check if the bead is already on the change list (type is negative). If not, the bead is copied intact, placed on the list, and its type set negative to indicate that it has been changed.
- DZRST: Go through the change file, restoring the changed beads to their original state. DZRST invokes a screen erase because deletes may have been performed on items which would not be captured correctly on the display. All types are set positive again.
- DZSAVE: Save the changed data base by eliminating the change list and setting all types positive. The entire data base must be scanned because a new bead may have been added to replace one that was too short, a fact not reflected on the change list.

Two utilities scan the attribute list for a particular attribute currently active in a data base. Both return the location in the attribute list of the stored value if present and zero if the attribute is not present. The first utility returns immediately, the second will create a new slot if the attribute has not been defined and then returned. A third utility checks if an element type is legitimate.

- IFATT: Fetch the location of the alphanumeric attribute. If the attribute is not in the list, return zero. If the attribute is active, return its location. If the attribute is not active and is on the list, define a location for it in this data base and return the new location. If the new location overflows MAXATT, create a new page for the attribute list.
- ISACT: Scan the active attribute list of the level for the name given.
- CNELM: Look at the attribute types to insure that the parameter

passed is a legitimate element type. Zero means an invalid type, else the element number.

ISACT is the more commonly used routine. IFATT is used internally to allow a change to be made to any attribute via DBCHA even though the attribute was not originally specified in a conversion routine. Efficiency dictates that this feature of DBCHA be used sparingly and all needed attributes be defined in the initial conversion routine.

The following three routines maintain the active lists for displayed beads on each level.

ACTIVA: Activate a bead on the proper level for display if the bead is not active already. The routine modifies the appropriate pointers and generates a new list if previously empty.

DELETE Deactive (remove from active list) and erase the picture from the screen.

ICKACT: Check if bead is active. The current version merely checks if an active right pointer (DBDRT) is present.

Specific low-level routines are available to set and retrieve information from the table. The actual formatting of the tables is done in conversion routine CNTABL, so the only specific setting is on a single word basis via:

DBTGET: returns the value of the (I,J,K) entry in the table.

Entry point

DBTSET: set the (I,J,K) value in the bead to the input value

Individual fields can be retrieved by:

IDBTFM: retrieve format of the table

Entry points

IDBROW: retrieve row dimension

IDBCOL: retrieve column dimension

IDBDEP: retrieve depth dimension

IDBFTY: retrieve format type (A or numeric)

IDBNDM: retrieve the number of subscripts.

2.5.5.2 Higher Level Data Base Routines

The routines in this section have been written specifically for Display and Edit. Their function is easily generalized to other modules. All deal directly with the data base and make calls directly to data base handler routines.

- GETCEN: Determine the center of an element by summing the node coordinates and dividing the results by the number of nodes. The value is returned in XC, YC, ZC in LIMIT.
- GETCES: Determine center of a structure by averaging the display space filled by a substructure. Value is in XC, YC, ZC of LIMIT.
- GETCOR: Get the coordinate of this node. The value is converted to cartesian coordinates, displaced if retrieving for a deformed plot, and checked against the current mins and maxes (CMIN and CMAX in LIMIT) to be used by the 'FILL SPACE' option. The coordinate locations are pointed to by IATLOC in LIMIT. The coordinate is returned in X, Y, Z of LIMIT. If shrink mode is enable, the X, Y, Z coordinate is returned properly as long a GETCEN was called first.
- GETCSS: Get center of substructure by averaging the display space. The routine is currently unused in Display and Edit.
- GETLIM: Get the limits of the display from any substructure bead. The limits are set to the largest space filled by the model. The values are returned in the parameter list.
- ELNCHK: Check if the new X, Y, Z of a node is outside the display space limits. If so, update the limits in the substructure beads. The input is X, Y, Z in LIMIT.
- ICKATT: Check if the values in a bead are within range (SVALL(NAMES(I)) to SVALH(NAMES(I)), I=1,IST. All variables are in common EDITT and set in text editing ICKATT is returned zero if not within tolerances.
- ICKRNG: Check if a type-in value is in the data base. If a range

is typed-in, the nearest neighbors are returned to avoid confusion. Zero indicates a single number that was not found.

ICKSL: Check if a displacement slice is active in the data base. Zero indicates slice not present.

NEXTIN: Traversing the data base can be done in its entirety or by substructure. NEXTIN looks at the number of active substructures and traverses the level properly. If NEXTIN=0, nothing is on the level. Entry NEXTB gets the next bead in the chain until the list is exhausted. When done, the function returns zero. Any data base level can be traversed by setting IACTYP in PERMEN to the level number and performing the following sequence:

```
      IB=NEXTIN(0)
10    IF (IB .EQ. 0) (Done)
      .
      . work on IB.
      .
      IB=NEXT(0)
      GO TO 10
```

If there are substructures active (NAC(2) greater than 0 in COMDECK ACTSTR), all elements or nodes are looked at for those substructures only. Otherwise, all the beads on that level are traversed.

2.5.5.3 Conversion Routines

The main purpose of STAGING is to permit a user to interact with finite element data stored in the general data base. Several subroutines have been generated to aid in moving information to and from the general data base. Three different programs must be written when adding a new analysis program to the STAGING system. Figure 2.5.11 illustrates the flow of information.

The initial data base for a model is constructed by conversion program 1. The data may then be viewed and corrected using STAGING. When

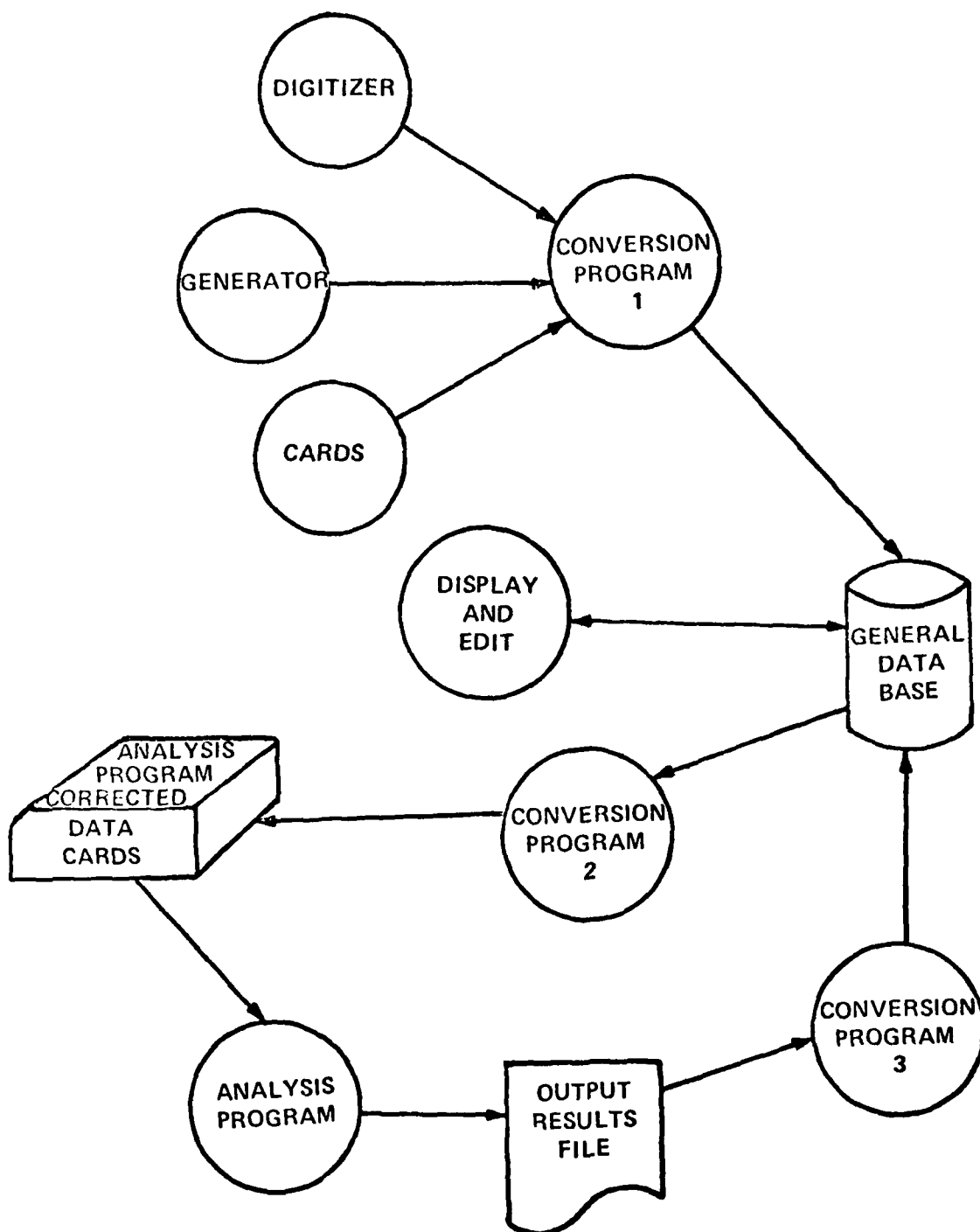


FIGURE 2.5.11 - INFORMATION FLOW

the engineer is satisfied with the model, conversion program 2 is used to retrieve data from the general data base and write a formatted input file for processing by the analysis program. The analysis program is executed in the batch environment yielding the output results data. Conversion program 3 then reads the resultant output data and adds it to the existing general data base file. STAGING may then be used to aid in the interpretation of the results. Please note the closed circle of programs which invokes the general data base, conversion programs 2 and 3, and the analysis program. The engineer may iterate as many times as is necessary to obtain a satisfactory solution.

High-level user-oriented subroutines have been developed to aid in the generation of conversion routines. Each of the three categories is described and example programs are provided. A "cook book" approach is recommended--start with the sample program and modify it.

When errors are detected (i.e., a node is attached to a non-existent element) an error message will be issued on the output file and processing will continue. The user may then use the Display and Edit module to correct the errors.

To reduce the programming effort, a set of utility subroutines have been generated. They provide for the initialization of the data base, creation of information at each of the four levels and completion of the data base. CNINIT, the initialization routine must be executed before any other routines are called. CNTERM, the completion routine, must be the last routine called. The other subroutines CNSTR, CNSUB, CNELEM, CNODE, and CNTABL, may be called in the order most convenient to the user.

CNINIT - Initialize the Data Handler data base, define and save active attributes and (CNINI2) set up internal arrays. Set up attribute entries for CNINIT. Tables 1, 2, and 3 list the attributes that are currently available in the STAGING system. New attributes will be added as they are identified. The list is quite long and most applications require only a small subset of the possible values.

Table 2.5.2 lists the element attributes currently defined in the system. The full name, the 10-character attribute name and an abbreviation are given. When you request an attribute display using Display and Edit, the abbreviations will be presented as identifiers.

The first 7 characters of each attribute name are unique. A

TABLE 2.5.2 ELEMENT ATTRIBUTES

No.	Full Name	Attribute Name	Abbreviation
1	Type	TYPE *	TYP
2	Material Identifier	MATERIAL	MAT
3	Area Cross-Section	AREA-CRSST	CSA
4	Area Moment X**Direction	X AREA MOM	XAM
5	Area Moment Y Direction	Y AREA MOM	YAM
6	Area Moment Z Direction	Z AREA MOM	ZAM
7	Torsional Constant	TORSIONAL	TOR
8	Mass/Length	MASS/LENGT	MPL
9	Membrane Thickness	MEM THICK	MTH
10	Mass/Area	MASS/AREA	MPA
11	Flexural Thickness	FLEX THICK	FTH
12	Material Property A	MAT-PROP-A	PRA
13	Pressure	PRESSURE	PRE
14	Temperature	TEMPERATUR	TEM
15	Critical Load	CRIT LOAD	CLD
16	Design Criterion	DES CRIT	DCR
17	Construction Code	CONSTRCODE	CCD
18	Geometry Class	GEOMCLASS	GCL
19	Geometry Sub-Class	SGEOMCLASS	SGC
20	Angle Between Prop-Axes & Side I-T	BETA	BET
21	Tension Allowable Stress	TEN ALWSTR	TAL
22	Compression Allowable Stress	CMP ALWSTR	SAL
23	Shear Allowable Stress	SHR ALWSTR	SAL
24	Minimum Size	MIN SIZE	MIN
25	Maximum Size	MAX SIZE	MAX
26	Allowable Class	ALLOWCLASS	ALC
27	Allowable Sub-Class	SALLOWCLASS	SCN
28	Average Stress Concentration Ratio	STRCNSTR	STC
29	Original Thickness	ORIG THICK	OTH
30	Excluded Element	EXCLUD ELM	EXE
31	Non-Optimum Weight Factor	NOPTWTFAC	NPW
32	Normal Stress X (Centroid)	X N STRESS	XNS
33	Normal Stress Y (Centroid)	Y N STRESS	YNS
34	Normal Stress Z (Centroid)	Z N STRESS	ZNS
35	Shear Stress X (Centroid)	X S STRESS	XSS
36	Shear Stress Y (Centroid)	Y S STRESS	YSS
37	Shear Stress Z (Centroid)	Z S STRESS	ZSS
38	Maximum Principal Stress	MAX STRESS	MXS
39	Intermediate Principal Stress	INT STRESS	INS
40	Minimum Principal Stress	MIN STRESS	MNS
41	Equivalent Stress	EQU STRESS	EQS

*See Table 2.5.3.

**XYZ - GLOBAL Cartesian Coordinate System

subscripting capability, not unlike that in FORTRAN, has been provided to allow any node or element attribute to assume multiple values. For example, a time dependent analysis may wish to define 5 different normal stress (X N STRESS) values. This would be accomplished by specifying a 10-character field, the first 7 characters of which are the first 7 characters of the attribute name (X N STR) followed by up to 3 digits defining subscript. For example:

```
IATELM(1) = 10HX N STR 1
      (2) = 10HX N STR 2
      (3) = 10HX N STR 3
      (4) = 10HX N STR 4
      (5) = 10HX N STR 5
```

(Note: The parameters 10HX N STRESS, 10HX N STR 0, and 10HX N STR 1 all refer to the same attributes.)

Table 2.5.3 lists the element types available in STAGING. Table 2.5.4 describes the NODE attributes. When CNINIT is called, it will reserve space for each attribute activated. New attributes may be activated at any time. However, the most efficient method of operation is to activate as many attributes as possible in the initial conversion routine. If there is indecision on about whether an attribute will be used, it is better to leave the attribute undefined until later.

CALLING PARAMETERS:

CALL CNINIT (IATNOD, NATNOD, IATELM, NATELM)

where:

IATNOD - NODE attribute array

NATNOD - Number of NODE attributes defined

IATELM - ELEMENT attribute array

NATELM - Number of ELEMENT attributes defined

The conversion routine CNINIT uses CNINI2 to specify both the attributes to be used and their relative locations. This is accomplished by using two parameters for the two lower levels in the data base. For example, NATELM specifies the total number of active element attributes. The order in which names are provided fixes their relative locations for subsequent calls to CNELEM (create an element item) and also in the general data base.

To elaborate, assume the user wants to specify MATERIAL, TYPE,

TABLE 2.5.3. ELEMENT TYPE VALUES ARE EITHER THE NUMBERS
(IN FLOATING POINT) OR SHORTHAND NAME

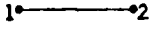
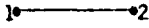
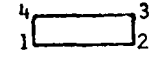
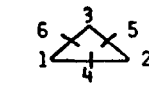
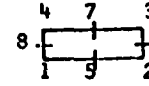
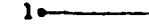
TYPE NO.		FULL NAME	SHORTHAND NAME
1		Rod	ROD
2		Straight Beam	STR BEAM
3		Membrane Triangle	MEM TRIA
4		Membrane Quadrilateral	MEM QUAD
5	Same as 3	Plate Triangle	PLATE TRIA
6	Same as 4	Plate Quadrilateral	PLATE QUAD
7	Same as 3	General Triangle	GEN TRIA
8	Same as 4	General Quadrilateral	GEN QUAD
9	Same as 3	Ring Triangle	RING TRIA
10	Same as 4	Ring Quadrilateral	RING QUAD
11	Same as 4	Shear Panel	SHER PANEL
12	Same as 4	Twist Panel	TWIS PANEL
13		General Triangle(2)	GEN TRIA2
14		General Quadrilateral(2)	GEN QUAD2
15	Same as 13	Ring Triangle(2)	RING TRIA2
16	Same as 14	Ring Quadrilateral(2)	RING QUAD2
17		Curved Beam	CURVE BEAM
18	Same as 17	Ring Shell	RING SHELL
19		Ring Conical	RING CNICL

TABLE 2.5.3. ELEMENT TYPE VALUES ARE EITHER THE NUMBERS (IN FLOATING POINT) OR SHORTHAND NAME (CONTINUED)

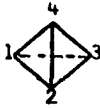
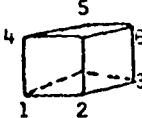
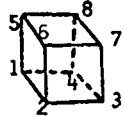
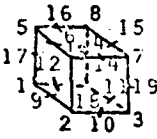
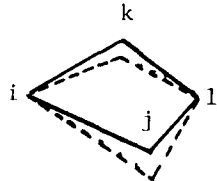
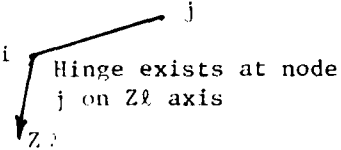
TYPE NO.		FULL NAME	SHORTHAND NAME
20		Tetrahedral Solid	TETR SOLID
21		Wedge Solid	WEDG SOLID
22		Hexahedral Solid	HEXA SOLID
23		20 Node Brick.	BRICK20
24	1—AS—2	Axial Spring	AXIAL SPRG
25	1—TS—2	Torsional Spring	TORSN SPRG
26	1—M—2	Mass	MASS
27	1—D—2	Damper	DAMPER
28	 warped quadrilateral	Warped Quadrilateral	WARP QUAD
29		Hinged Beam	HINGE BEAM

TABLE 2.5.4 NODE ATTRIBUTES

No.	Full Name	Attributes	Abbreviation	Values
1	X Coordinate	X-CORD	X	any ¹
2	Y Coordinate	Y-CORD	Y	any ¹
3	Z Coordinate	Z-CORD	Z	any ¹
4	Coordinate System	CORD-SYSTEM	COS	1=Cartesian 2=Polar(2D)Cylindrical(3D) 3=Spherical(3D Only)
5	X Force	X-FORCE	XFO	any
6	Y Force	Y-FORCE	YFO	any
7	Z Force	Z-FORCE	ZFO	any
8	X Moment	X-MOMENT	XMO	any
9	Y Moment	Y-MOMENT	YMO	any
10	Z Moment	Z-MOMENT	ZMO	any
11	Force Coordinate System	FORCE-CORD	FCO	Local Cord Sys. ID
12	Balance Weight	BAL WT	BWT	any
13	X Rotation	X-ROTATION	XRO	any
14	Y Rotation	Y-ROTATION	YRO	any
15	Z Rotation	Z-ROTATION	ZRO	any
16	Displacement Coordinate System	DISP-COORD	DCO	Local Coord Sys. ID
17	Temperature	TEMPERATUR	TEM	any

TABLE 2.5.4 (Continued)

No.	Full Name	Attributes	Abbreviation	Values
18	Concentrated Mass	CONC-MASS	CMS	any
19	Pressure	PRESSURE	PRE	any
20	X Displacement	X DISP	XDS	any ²
21	Y Displacement	Y DISP	YDS	any ²
22	Z Displacement	Z DISP	ZDS	any ²
23	X Mode Shape	X MODE	XMD	any ²
24	Y Mode Shape	Y MODE	YMD	any ²
25	Z Mode Shape	Z MODE	ZMD	any ²
26	X Load	X LOAD	XLD	any ²
27	Y Load	Y LOAD	YLD	any ²
28	Z Load	Z LOAD	ZLD	any ²

¹ Interpretation of X, Y, Z varies according to Attribute 4. If CORD-SYSTM is 1.0 (Cartesian), X, Y, Z are coordinates in space. If CORD-SYSTM is 2.0 and only X and Y are provided (Polar), Attribute 1 is R and Attribute 2 θ measured in radians. If X, Y, Z are provided (cylindrical), Attribute 3 is Z as in the Cartesian case. If X, Y, Z are provided and CORD-SYSTM is 3.0 (spherical), Attribute 1 is R, 2, θ (radians) and 3, θ (radians).

² The displacement factors are delta displacements from the node Point X, Y, A and must be provided in the same coordinate system. The delta values can refer to up to ten mode shapes, load conditions, or time steps. Only one of these conditions (mode shapes, load conditions, or time steps) can be active at any one time in the data base.

PRESSURE, TEMPERATURE, and MEMBRANE THICKNESS as element attributes. Remember this will fix the order in which they will be provided in subsequent calls to CNELEM. Referring to Table 1 we find the appropriate attribute names. To accomplish this goal, the parameters NATELM, IATELM supplied to the initialization routine CNINIT would contain the following values.

```
NATELM      = 5 (5 attributes activated)
IATELM (1)  = 10HMATERIAL
IATELM (2)  = 10HTYPE
IATELM (3)  = 10HPRESSURE
IATELM (4)  = 10HTEMPERATURE
IATELM (5)  = 10HMEMTHICK
```

CNSTR --Create structure bead with a 40-character name and store substructure linkage into general data base.

CALLING PARAMETERS:

CALL CNSTR (NUM, SUBS, NSUB)

where:

```
NUM - Substructure name (40 character)
SUBS - Array containing the names of substructures that belong
       to this structure (4 words per name)
NSUB - Number of substructure names supplied in SUBS
```

CNSUB --Create substructure bead with a 40-character name and store element linkage into general data base.

CALLING PARAMETERS:

CALL CNSUB (NUM, IELEM, NELEM)

where:

```
NUM - Substructure name (40 characters)
IELEM - Array containing the element numbers that belong to
         this substructure (integer format)
NELEM - Number of element names supplied in IELEM.
```

CNELEM--Create element with positive integer name, store attributes and node linkage into general data base.

CALLING PARAMETERS:

CALL CNELEM (NUM, NODE, NMODE, ATT, NATT)

where:

NUM - Element name (1 word; integer format)
NODE - Array containing the node numbers that belong to
this element (integer format)
NMODE - Number of nodes supplies in NODE
ATT - Attribute array
NATT - Number of attributes supplies
Elements must supply the TYPE attribute.

CNNODE--Create node with positive integer name and store attributes.

CALLING PARAMETERS:

CALL CNNODE (NUM, ATT, NATT)

where:

NUM - Node Number (integer format)
ATT - Attribute array (optional)
NATT - Number of attributes supplies (optional)

CNTABL--The CNELEM and CNNODE routines allow the user to store element and node data into the general data base. The subroutine CNTABL supports the storage, retrieval, and revision of tabular information. The user provides a unique name for the block of data, its format and dimensionality. This subroutine is used by the conversion routines to store data not associated with elements or nodes, i.e., material properties, analysis code options, etc.

CALL PARAMETERS:

CALL CNTABL (NAME,ARRAY,NR,NC,ND,FORMAT)

where:

NAME - Block data name (40 characters)
ARRAY - Array containing the dataa to be stored
NR - Row Dimension
NC - Column Dimension
ND - Depth Dimension
FORMAT- Up to 30 characters FORTRAN format specification
(i.e., 4H(I5) Default is 4H8A10).
NOTE - For a single dimensioned array the call can be reduced
to:
CALL CNTABL (NAME,ARRAY,NR,FORMAT)

Since subscripting is done ala FORTRAN (storage by row), the dimensions of the array for a 2 or 3 dimension table must be the same as the dimensions passed to CNTABL. This essentially implies that the column (in the 2D case) and column and depth (in the 3D case) be exactly the same as the dimensions indicated in the DIMENSION statement to insure correct retrieval. New tables can be added during any conversion phase with CNTABL. Certain functions in these routines have been broken into separate subroutines to make the larger routines easier to follow.

They are:

CNINI2: Sets up attribute entries for CNINIT.

ICPART: It is possible for the user's list of down pointers and attributes to overflow the internal scratch array. Therefore, routine ICPART will flush the array before it is overflowed and restart the filling process.

CNTER : The most time consuming part of conversion is inserting the up and down pointers. The down pointers are the only ones provided, and they are names rather than bead address. Therefore, CNTERN queues the down pointers names and sorts them in CNTER to minimize search times before setting the up pointers. The list is then re-sorted to assign the bead addresses of the down pointers. N.B. if there is an error in the list of down pointers, no down pointers will be assigned.

CNSORT: Does the sort for CNTER as a hash sort on the low order 30-bits of the word.

Three utilities are present to print out features of the data base. These are:

CNPRT : Print a bead and its up and down pointers.

CNPRTA: Print a bead and its attributes.

DPRT : Dump the data base and attribute arrays.

Additional conversion routines have been coded to speed the coding process for conversion phases two and three. These routines do not support addition of new beads to the data base. It is assumed that those tasks will be accomplished in preprocessors of Display and Edit.

*****--Complete conversion process.

CALLING SEQUENCE:

CALL CINTERM

This routine consumes 75% of the conversion processing. It first places into ascending order the out-of-order elements and nodes, sets up the ILKUP array, assigns all up and down pointers (probably the most time consuming process--refer to CINTER and CNSORT for more details), generates a default structure and substructure if no CNSTR or CNSUB calls were made, assigns the plot limits (saved by CNNODE) to all substructures, flushes the proper arrays (RESTART), and reinitializes the EDITT COMDECK (EDITTI).

2.5.5.3.2 Conversion Program 2

Once the user has corrected and optimized his model using Display and Edit, he will be ready to execute an analysis program. Conversion Program 2 will be used to transfer information from the general data base to a card image file suitable for processing the analysis program. Conversion Program 2 is the exact inverse of Conversion Program 1. The utility subroutines used are similar in appearance and structure.

NCINIT - Initialize the conversion programs and define the attributes that are to be retrieved from the data base. The method of definition and handling of parameters for Conversion Program 2 is identical to Conversion Program 1. Since attributes cannot be retrieved from the general data base that have not been created, we recommend repeating the initialization code used by Program 1 in Program 2. Note the order in which you specify the attributes is the order in which they will be returned to you. This order does not have to be the same as in conversion routine 1. If a new attribute is mentioned, space will be assigned automatically.

CALLED PARAMETERS:

CALL NCINIT (IATNOD, NATNOD, IATELM, NATELM)

where:

IATNOD - Node attribute array

NATNOD - Number of NODE attributes activated

IATELM - Element attribute array

NATELM - Number of ELEMENENT attributes activated

NCINIT uses

NCSTUF: set up list of attributes for this level. The user can then retrieve information from a data base in one of two ways. (All routines listed are entry points into subprogram NCELEM).

1. Work on the entire list on a level. In this case, the get routines return the entity names as well as attributes:

NCSTR : get next structure name (40 characters), attributes, (40 characters each) all substructures names and number of substructures.

NCSUB : get next substructure name (40 characters), attributes, all element numbers (integers), and number of elements.

NCELEM: get next element number, attributes, all node numbers, (integers), and number of nodes.

NCNODE: get next node number and attributes. These routines return a 0 for the next number if the list is exhausted. Thus, the loop

```
10 CALL NCELEM(NUM,ATT,NODE,NNODE)
   IF (NUM .EQ. 0)   Done
   do something here
   GO TO 10
```

will loop through all elements.

2. Each entity can be worked on individually. This assumes that the programmer knows the names he is after. The routines return the same information as (1).

NCGSTR: get info from given structure

NCGSUB: get info from given substructure

NCGELM: get info from given element

NCGNOD: get info from given node

The user can reset the attributes. The "do something here" portion of the above example will often manipulate information in the attributes array and then replace that information by:

NCSSTR: reset attributes in given structure

NCSSUB: reset attributes in given substructure

NCSELM: reset attributes in given element

NCSNOD: reset attributes in given node.

NCSTR --Retrieve structure and substructure linkage from general data base.

CALLING PARAMETERS:

CALL NCSTR (NUM, ATT, SUBS, NSUB)

where:

NUM - Structure name (40 characters or 4 words) NOTE: Each call to NCSTR will return a new structure name until the list has been exhausted, then NUM(1) will be set to zero.

ATT - Dummy parameter

SUBS - Array of names of substructures attached to this structure (4 words per name)

NSUB - Number of substructure names supplied in SUBS

NCSUB --Retrieve substructure and element linkage from the general data base.

CALLING PARAMETERS:

CALL NCSUB (NUM, ATT, IELEM, NELEM)

where:

NUM - Substructure name (40 characters or 4 words) Note: each call to NCSUB will return a new substructure name until the list has been exhausted, then NUM will be set to zero.

ATT - Dummy parameter

IELEM- Array of element numbers attached to this substructure (Integer format)

NELEM- Number of element numbers supplied in IELEM

NCELEM--Retrieve elements, attributes and modes linkage from general data base.

CALLING PARAMETERS:

CALL NCELEM (IELEM, ATT, NODE, NNODE)

where:

IELEM- Element number (integer format) Note: Each call to NCELEM will return a new element number until the lists is exhausted, then IELEM will set to zero. The element numbers are returned in ascending order.

ATT - Element attribute array

NODE - Array of node numbers attached to this element (integer format)

NNODE- Number of nodes numbers supplied in NODE.

NCNODE--Retrieve nodes and attributes from general data base.

CALLING PARAMETERS:

CALL NCNODE (NODE, ATT)

where:

NODE - Node number (Integer format) Note: Each call to NCNODE will return a new NODE number until the list is exhausted, then NODE will be set to zero. The node numbers are returned in ascending order.

ATT - Node attribute array

To get or reset information in an existing table:

NCTABS: Set the (I,J,K) element in the table name to a new value
Entry points

 XNCTABG: Get the (I,J,K) element in the table

 NCROW : Get the row dimension (set by CNTABL)

 NCCOL : Get the column dimension (set by CNTABL)

 NCDEP : Get the depth dimension (set by CNTABL)

 NCFORM : Get the format of the table (set by CNTABL)

The format specification appears mysterious at first glance. The format is defined as a variable format in FORTRAN as a Hollerith string. It serves a twofold purpose. First, a set of conversion routines can set up various table formats as part of the table itself. This can let different input formats be accommodated with variable formats. The second use is internal. Tables are most convenient as application dependent storage, in which case the data is of A format. However, in the X-Y plot package in Display and Edit, one does not really want to draw data cards. Therefore, the format also determines if the table is plottable or not.

An A format says do not plot, else the values are legitimate. The user must beware if using integer values because the "get" routines are real unless declared integer.

NCASU --Activate substructure for retrieval. If the engineer using Display and Edit has created several substructures and wishes to perform an analysis on specific substructures, NCASU provides that capability. If NCASU is called, only the elements and nodes belonging to the listed substructures will be available to the engineer in subsequent calls to NCELEM and NCNODE. If NCASU is not called, all information in the general data base will be available to the user. Note NCASU may be called at anytime after NCINIT and may be called as often as desired. However, only those substructures listed in the latest call will be active.

CALLING PARAMETERS:

CALL NCASU (SUBS, NSUBS)

where:

SUBS - Substructure names (40 characters or 4 words each)

NSUBS - Number of substructure names supplied in SUBS

NUMC --Return the number of items within a level. This utility function is useful in obtaining information about the data base.

CALLING PARAMETERS:

NUM = NUMC (TYPE)

where:

ITYPE - data base level

= 1 - Structure

= 2 - Substructure

= 3 - Element

= 4 - Node

NUM - number of items present in data base

XNCTABG-Function NXCTABG is used to obtain block data stored by the subroutine NCTABL.

CALLED PARAMETERS:

VAL = XNCTABG(NAME,IR)

where:

NAME - 40-character name

IR - entry in array desired

VAL - data returned to user

NCFORM--Subroutine NCFORM is used to obtain the FORMAT stored for a specific block data table.

CALLING PARAMETERS:

CALL NCFORM(NAME,FORMAT)

where:

NAME - 40-character name

FORMAT - Format as supplied by user, up to 30 characters

NCTERM--The counterpart of CINTERM for the retrieval routines is NCTERM. This must be the last NC routine called.

CALLING PARAMETERS:

CALL NCTERM

Terminate conversion program two or three. The user's data base is automatically extended if it is a permanent file. WARNING: if the user is resetting information in conversion program two or three and his program bombs, his data base may be destroyed.

A control card safety play is:

ATTACH,T,data base.

REQUEST,TAPEO,*PF.

COPYBF,T,TAPEO.

Run conversion routine two or three.

PURGE,T.

CATALOG,TAPEO,data base.

2.5.5.3.3 Conversion Program 3

This is the last conversion program of the sequence. It is used to add the output results from an analysis to an existing data base. Another set of conversion subroutines have been generated to aid in this task. They are similar in concept and function to those used in conversion programs one and two. At each level in the data base they are used to add or modify information in an existing data base.

NCINIT--Initialize the conversion subroutines and identify the attributes you wish to add or modify in the data base. This subroutine is the same one described under Section 2.5.5.3.2. Now you will be specifying the order in which you will be supplying results data.

CALLING PARAMETERS:

CALL NCINIT (IATNOD, NATNOD, IATELM, NATELM)

where:

IATNOD - Node attribute array

NATNOD - Number of node attributes activated

IATELM - Element attribute array

NATELM - Number of element attributes activated

NCSELM--Set element attributes.

CALLING PARAMETERS:

CALL NCSELM (NUMBER, ATT)

where:

NUMBER - element number (integer format)

ATT - attribute array

NCSNOD--Set node attributes.

CALLING PARAMETERS:

CALL NCSNOD (NUMBER,ATT)

where:

NUMBER - node number (integer format)

ATT - attribute array

NCTABS--Set a value in a table.

CALL NCTABS (NAME,VAL,IR)

where:

NAME is the 40-character name of the table

VAL is the new value to insert into the array

NCTERM--Terminate the conversion subroutines and extend the permanent file
TAPE0.

CALLING PARAMETERS:

CALL NCTERM

2.5.5.4 Initialization and Termination

All of the features of each individual data base bead are maintained in core resident tables. These tables are also stored in the user's data base and are restored during file initialization. Termination of a user session requires the same arrays to be saved before the data

handler file is flushed. If no changes are made to the data base during a run, there is no need to flush the arrays before exiting.

Subroutines used here are:

- DBINIT - Initialize the file name provided by reading in the arrays stored by RESTART. A change file is initiated if desired, otherwise overflowed beads are released. If the file is not a proper data base file, DBINIT returns zero.
- RESTART - Store significant arrays in this data base. The routine checks if the file has been used before and releases existing space if possible and calls DMFLSH when done.
- GLOINT - Contains DATA statements to set up initial guesses for all attribute array sizes in this data base. These values can be changed before RESTART is called to make increases in size. Most common is to increase the size of NO MAN'S LAND in each bead (JDEF array in COMDECK ATTRIB). Also contains DATA statements for data base file and element names.
- EDITTI - Initialize the EDITT common block which is used as a scratch common block by other modules (most notably the conversion routines).
- FILATT - Checks type-in for a good permanent file name and tries to attach it. If the file is (a) not catalogued, (b) illegally typed, or (c) an illegal data handler file, FILNAM is called again to retry the type in. If the type-in is a success, the file is attached and copied. DBINIT initializes the file and DEINIT1 initializes the proper Display and Edit variables.

2.6 INTERTEK Interactive Graphics Package

2.6.1 Introduction

The "display file" of an interactive computer graphics system consists of:

1. An ordered set of instructions for displaying graphic entities on a physical display device.
2. A set of identifiers and attributes associated with the graphic entities in the display file.

A graphic entity or segment is a collection of graphic primitives such as dots, vectors, and alphanumeric characters, which are typically generated by the display device.

INTERTEK is a series of FTN subprograms call-modeled on 777/IGS V2.1 and designed to enhance the interactive capability of a Tektronix 4010 Series Direct View Storage Tube. INTERTEK runs on CDC 6000 series mainframes under NOS/BE. It is written for the CDC FTN compiler and uses the NSRDC Data Handler, Battelle developed file manipulation utilities, and the graphics driver COMPIO.

2.6.2 INTERTEK Picture Manipulation

INTERTEK routines handle display file construction and modification, as well as picture scaling, translating, and clipping. It also supports general pick processing of pictures on the terminal screen.

The basic drawing information is contained in the STAGING display file as a data structure with hierarchy illustrated in Figure 2.6.1. The three major components of the display file are:

1. Display item
2. Subfile
3. Display area entity

A display item is the basic picture building block. It is the smallest unit which can be created, modified, or picked. Each display item contains the display commands for generating a graphic entity as well as control information for pick processing. The display items in the STAGING display file are maintained on Data Handler file DISFILE.

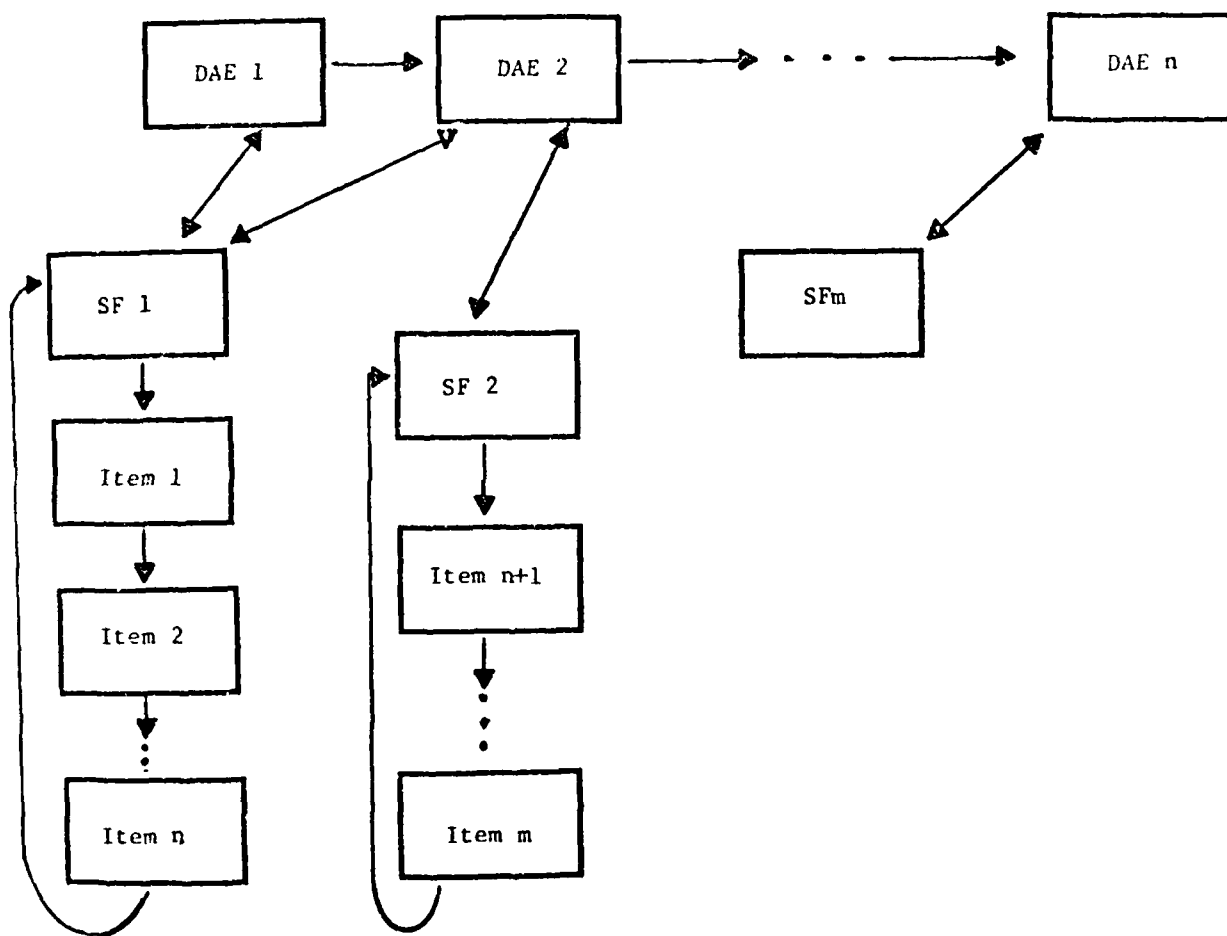


FIGURE 2.6.1 - DISPLAY FILE CONSTRUCTION

A subfile is a linear linked list of display items headed by a subfile entity which is maintained as a 6-word block in array IARIA in common block GCGI. Items of a subfile may be manipulated collectively.

A display area entity (DAE) represents a complete virtual picture. It contains information defining the display device coordinates of the rectangular boundary of the virtual picture. The DAE also contains information defining which of its subfiles are currently visible. A subfile may be owned by more than one DAE. Thus a picture can be displayed in several areas on the terminal screen. This capability facilitates the split-screen function in the STAGING system. Each DAE in STAGING is maintained as a 6-word block in array IAREA in common block GCGI.

The formats of the DAE's and subfile entities in INTERTEK are shown in Figures 2.6.2 and 2.6.3. The display item Data Handler bead format is illustrated in Figure 2.6.4. The actual display information within a display item consists of instructions in the formats shown in Figure 2.6.5.

An item is created by calls to INTERTEK subroutines with prefix GU. Each call to one of these routines causes drawing information to be packed into array IBUF in COMMON block DRAWBUF (external to the INTERTEK package). The format of IBUF is shown in Figure 2.6.6. When the program reaches a full IBUF or the end of a display item, GITEM is called to generate a disk copy and to generate the first drawing of the picture. If the item overflows IBUF, the user calls GITEM, packs more information into IBUF, and calls GIXTND to extend the item. The result is a chain of beads on the Data Handler file that comprise the whole item. The format of an extension bead is shown in Figure 2.6.7. The bead list is terminated by an extend bead set to zero.

2.6.3 INTERTEK Pick Processing

Pick processing is controlled by INTERTEK by user definable assignment of event types to categories. There are five event types in INTERTEK:

button

DAE format

Word	59	44	29	14	0
1	Type = 1				
2	X min clip	Y min clip	X max clip	Y max clip	
3	(XWC) _X window center		(YWC) _Y window center		
4	(SCALE) scale factor (zoom level)				
5	Subfile 1	Subfile 2	SF 3	SF 4	SF 5
6	SF 6	SF 7	SF	SF 9	Subfile 10

where Type is the entity type

Clipping limits for X',Y' are in screen coordinates
 (-2048 < val < 2048).

Each coordinate is scaled and centered by $X' = X/Scale + XWC$
 $Y' = Y/Scale + YWC$

Sf1...Sfn are the subfiles shown in the area

FIGURE 2.6.2 - DAE FORMAT

Subfile Format

1	Type = 2									
2	left item pointer					right item pointer				
3	Cat 1	Cat 2								
4										Cat 20
5	DAE 1		DAE 2							
6									DAE 10	

where Type is the entity type

Left pointer is the last item in the subfile (if empty, this is the subfile)

Right pointer is the first item in the subfile (if empty, this is the subfile)

Cat 1...Cat 20 are the categories to which the item in the subfile belong

DAE 1...DAE 10 are the DAE's in which this subfile is shown

FIGURE 2.6.3 - SUBFILE FORMAT

AD-A089 382 RATTLE COLUMBUS LABS OH

F/6 13/13

SEP 79 L E HULBERT, N D GHADIALI, F N DEOBOT F33615-76-C-3125

AFFDL-TR-79-3074-VOL-3

NIL

AD
AD=9.59e7

1

END

DATE _____

10-12

10

1	Subfile owner of (SF) Item		Length of bead (LEN)				
2	bead address of extension (EXTB)						
3	left item pointer		right item pointer				
4	X MIN		Y MIN				
5	X MAX		Y MAX				
6	ITASKC						
LPROLG = 7	# of ID (LID) words	# of (LACT) categories	CAT 1	CAT 2	CAT 3	CAT 4	CAT 5
	ID 1 . . . ID n						if #ID words >0
LPROLC+n+1	absolute beam move drawing instructions						

where

SF is the subfile owner of this item (used only in return on a pick and when assigning the categories in a subfile).

LEN is the total length of the bead

EXTB is the bead address of the first extension. if=0, no extension.

Left-Right: preceding and succeeding items.

XMIN,YMIN,XMAX,YMAX: lower left and upper right corners of area containing item. Used when determining if item picked.

ITASKC: task word (user supplied)

LID and ID₁...ID_n: number of ID words (may be zero) and user supplied ID words

LCAT and CAT₁...CAT_n: number of pick categories and up to 5 user supplied categories

FIGURE 2.6.4 - ITEM FORMAT

Long Relative Vector (created by GULIN)

59					0				
OP	STY	B	REL X			REL Y			

FIELD	BITS	DESCRIPTION
OP	59-58	Opcode of 00 ₂ for long relative vector
STY	57-55	Linestyle (0=solid, 1=short dash, 2=long dash)
B	54	Beam (0=off, 1=on)
RELX	53-27	Relative X beam move
RELY	26-0	Relative Y beam move

Short Relative Vector (created by GULIN)

29					0				
OP	STY	B	REL X			REL Y			

FIELD	BITS	DESCRIPTION
OP	29-28	Opcode of 01 ₂ for short (halfword) relative vector
STY	27-25	Linestyle (0=solid, 1=short dash, 2=long dash)
B	24	Beam (0=off, 1=on)
RELX	23-12	Relative X beam move
RELY	11-0	Relative Y beam move

Absolute Vector (each item must begin with this instruction created by GUSETP)

59					0				
OP	P	Z	ABS X			ABS Y			

FIELD	BITS	DESCRIPTION
OP	59-58	Opcode of 10 ₂ for absolute vector
P	57	Pickable flag (0=nonpickable, 1=pickable)
Z	56	Zoomable flag (0=nonzoomable, 1=zoomable)
ABSX	53-27	Absolute X coordinate of item origin
ABSY	26-0	Absolute Y-coordinate of item origin

Text (created by GUTEXT)

59									
OP	SIZE	NC	CHI	CH2	 (may extend over multiple words)				

FIELD	BITS	DESCRIPTION
OP	59-58	Opcode of 11 ₂ for text output
SIZE	57-54	Character size from smallest (0) to largest (3)
NC	53-42	Number of characters in string
CHI	41-	Display code characters of string

FIGURE 2.6.5. DISPLAY ITEM GRAPHICS INSTRUCTION FORMATS

1	Current x (CX)	Current y (CY)
2	unused bit count (UBC)	
3	XMIN	YMIN
4	XMAX	YMAX
5	drawing instructions from GU-routines.	

MBYTE/3

where CX,CY is the current beam positions
 UBC is the number of bits left in the NBYTEth word
 XMIN, YMIN are the coordinates of the lower left
 corner of the item in the item's coordinate scale
 XMAX,YMAX are the coordinates of the upper
 right corner of the item

FIGURE 2.6.6 - IBUF FORMAT

Length
extension bead (EXTB)
Drawing instruction

where:

Length is the total bead length

EXTB is the next bead in an extension.

If EXTB=0, this is end of extension.

FIGURE 2.6.7 - EXTENSION FORMAT

ignore
single
string
parameter

Categories 0 to 63 may be assigned any of the above event types. Keyboard keys may be assigned to a single category. Pick processing is done by calling GIBUTN which turns on the crosshairs and waits for a pick. If the pick is a single pick, string pick, or parameter pick, an entry is made on the proper queue for later retrieval.

Each display item contains information about the action to be taken when the item is picked. The information consists of from 1 to 5 categories each of which is stored as an index into the category table in array ICATS in common block GCGI. The format of the category table is shown in Figure 2.6.8. Each entry in the table is a string of 6 bits indicating whether any of the following six actions are to be taken:

<u>BIT</u>	<u>ACTION</u>
1	Not implemented
2	Single pick
3	String pick
4	Parameter pick
5	Button pick
6	Not implemented

A button pick is the expected response to a call to subroutine GIBUTN. The pick of a selection from a STAGING menu is the typical use of a button pick. When pausing for a button pick, other types of picks may occur. Each non-button pick is placed on the queue corresponding to the indicated action. Single picks always replace what was there. String picks normally add the pick to the end of the string queue. However, if an occurrence of the pick already resides in the queue, it is deleted rather than added to the queue. Each parameter pick always adds a new entry to the end of the queue. Each of the three queues has a pointer to the head (IHLIST(I)) and a pointer to the tail (ITLIST(1)) as shown in Figure 2.6.9. Each queue entity holds information about the item selected in the format shown in Figure 2.6.10.

The programmer can also assign one category to each key on the

ICATS(1)	CAT 0	CAT 1	CAT 2		CAT 9
(2)					
(3)					
(4)					
(5)					
(6)					
(7)				CAT 63	

Each CAT Entry is a series of 6 bits

where

S _P _A	B _U _T	P _A _R	S _T _R	S _N _G	A _T _R
-----------------------------	-----------------------------	-----------------------------	-----------------------------	-----------------------------	-----------------------------

if all bits are zero, the event is an ignore. Otherwise, each bit indicates the type of action

ATR: attract (in this version, this is a no-op)

SNG: single pick

STR: string pick

PAR: parameter pick

BUT: button pick

SPA: special action (in this version, this is a no-op)

FIGURE 2.6.8 - CATEGORY TABLE

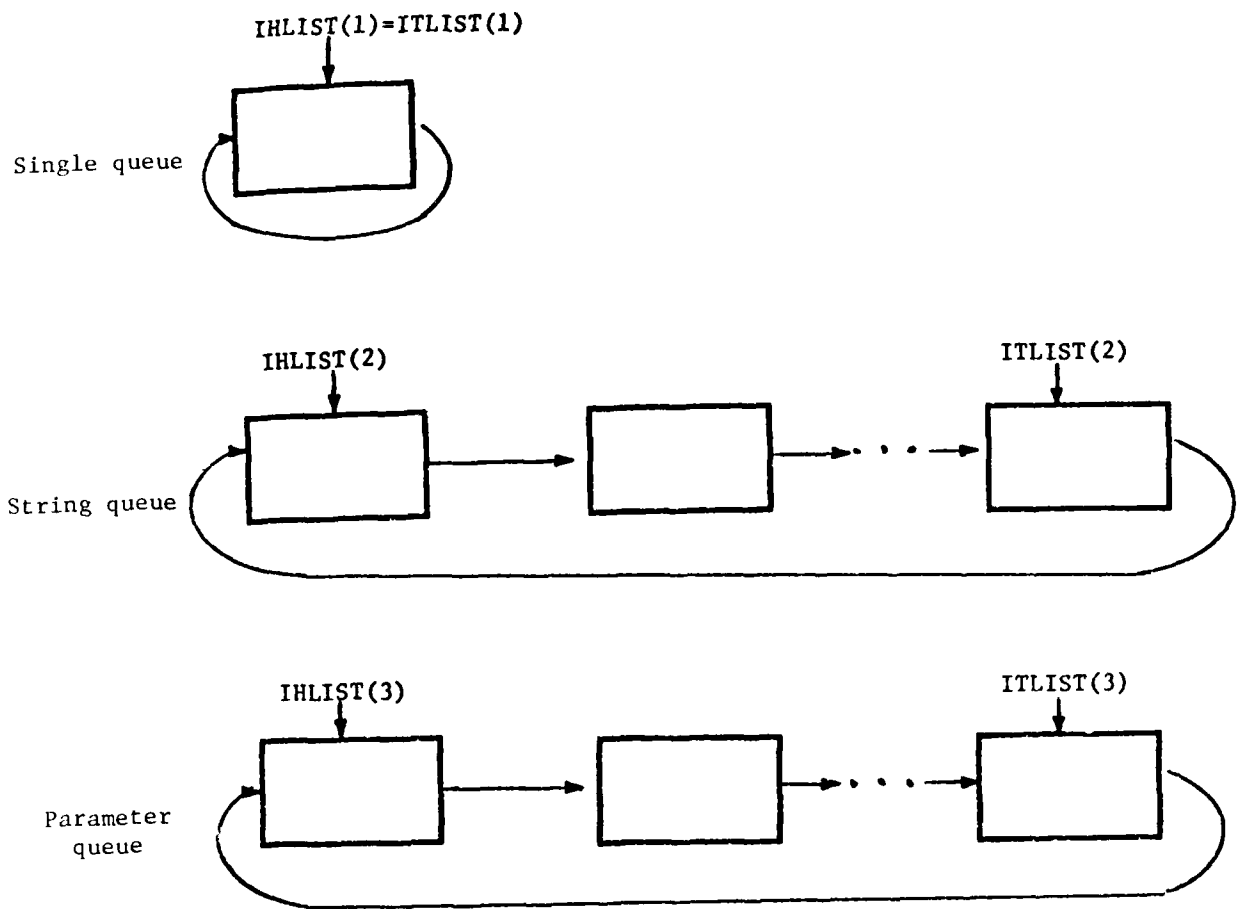


FIGURE 2.6.9. QUEUE CONSTRUCTION

1	Length of bead
2	Next bead in queue
3	ITASKC
4	IDDAD of picked item
5	SUBFILE owner of picked item
6	DAE in which picked item resides
7	X location of pick
8	Y location of pick
9	Character struck to generate pick
10	Number of categories
11	Number of ID words
12 + LCAT	Categories to which item belongs
12 + LCAT + 1	ID words of item (if any)

FIGURE 2.6.10. WAIT QUEUE ENTITIES

Tektronix keyboard. These categories are stored in array KEYCAT in COMMON block GCGI as shown in Figure 2.6.11. Only a button may be generated. Such a button pick is generated only if the crosshairs are not positioned on a displayed item.

2.6.4. INTERTEK Software Overview

INTERTEK is modular in construction. Six major functions have been identified:

- o initialization and termination
- o construction of picture
- o construction of display file
- o display file processing
- o event processing
- o utility routines

The routines in these areas rarely overlap in function. Instead, most are small and single purpose. This section will examine and classify INTERTEK routines, while briefly describing their functions.

2.6.4.1 Initialization and Termination

Initialization of two types occurs in INTERTEK. Before any other INTERTEK routine can be called, the programmer must call:

GINIT : set up INTERTEK. This includes all terminal dependent constants and the display file in its "empty" condition, i.e. only the default DAE and subfile are present

The user must supply a buffer as transient storage for graphics information. After calling GINIT, he must tell INTERTEK where and how long this buffer is:

GUBUF : define buffer for packing graphics information.

All non-transient data in common blocks is set up by a block data subroutine called GPRESET. There is no executable code in GPRESET. The application can leave a blank screen by calling:

GIRLS : return the display file and erase the screen.

KEYCAT (1)	CAT :	CAT A	CAT B	CAT C	CAT D	CAT E	CAT F	CAT G	CAT H	CAT I
(2)										
(3)										
(4)										
(5)										
(6)										
(7)		CAT ;								

Each key can have one category assigned to it and no ID words.

The categories are associated and detected if there is no pick on an item. Furthermore, only a button pick can be generated.

FIGURE 2.6.11. KEYBOARD EVENT TABLE

2.6.4.2 Picture Construction

The routines the user sees for constructing the picture are the GU-routines:

GUSETP : set up the absolute beam location, lightpen sensitivity, and zoomability

GULIN : move the beam to the new location with beam on or off

GUPNT : draw an * (there is no point generator on the Tektronix)

GUTEXT : put a text string on the screen.

As the buffer is packed, the GU-routines keep track of the last beam position and the rectangular area (lower left and upper right corners) containing the item. The area is used during pick processing to eliminate items from consideration quickly.

GCUPXY : updates the current beam position and rectangular area fields of IBUF.

Actual packing into and out of IBUF is handled by COMPASS routines for speed:

with entry points:

GCSET : fill the next available byte in IBUF

GCSETW : Entry point set the Nth word in IBUF

GCSETI : set a field in word N of IBUF for IL bits

GCGET : get bits from the 60 bit parameter (this routine is used as an unpack for all of INTERTEK)

with entry point:

GCGETW : Entry point get bits from the Nth word of IBUF.

Parameters to GCGET and GCSET are passed through common block GCBUF. It is the start bit of the get or set (60 -- 1), IL is the length of the string to get or set, ISEXT is a flag set non-zero if the result is to be sign extended (get only).

2.6.4.3 Display File Construction and Manipulation

Three basic entities comprise the display file. Each is created by a different GI-routine:

GIDAE : create a display area entity
with entry point:
GIDAE1 : reset zoom, window, and clip limits
GISUBF : create a subfile entity
GITEM : create a display item

Each routine merely stores the proper data items away. All use GCRESI to reserve a Data Handler bead or incore block of the information. The three are differentiated by a call to FUNCTION GCGTYP.

Items are more complicated to work with because they are Data Handler beads. The basic task of GITEM is to transfer the contents of IBUF into the bead. Special purpose routines are available to get and set information in the bead via Data Handler.

GCSSF : set the subfile to which the item belongs
Entry Points:
GCGSF : get the subfile of the item
GCGLN : get the length of the item
GCSLFT : set the left pointer of the subfile or item
GCGLFT : get the left pointer of the subfile or item
GCSRIT : set the right pointer of the subfile or item
GCGRIT : get the right pointer of the subfile or item
GCSONE : set the nth word of the item
GCGONE : get the nth word of the item
GCGLID : get the length of the ID block of the item

Bulk information (i.e., the contents of the displayable information) is passes into and out of the item by:

GCGARR : get an array of information from the item
with entry point:

GCSARR : Entry point set an array into an item

In addition to setting up information in the item, GITEM is responsible for updating the category list in the subfile. This category information is used at pick processing time to see if any of the items are in a

processable category. The update is done by:

GCUPSF : scan the subfile of the item and add any new categories to the list. Up to twenty categories can be in any one subfile.

An item can be extended. This is especially useful when IBUF would otherwise overflow:

GIXTND : extend the given item. (Note: extensions are valid only for the current working item; the next relative move depends on the beam position information in IBUF).

Linkage of a subfile and its item is done at item creation. Subfiles and DAE's are linked and unlinked by:

GISHOW : turn a subfile on or off in a DAE.

Two small routines are used to unpack and pack subfile and DAE arrays. These arrays are the cross referenced subfile-DAE lists and the subfile category list.

GCUNPK : unpack the list for a subfile or DAE into individual words

Entry Point:

GCUNPC : when subfile categories are being checked, they too need to be unpacked. GCUNPC accomplishes this task.

GCPKSD : Pack a new subfile or DAE into the proper location (used by GISHOW).

As the display file is constructed, the programmer can modify it in a number of ways. DAE's can be modified by:

GIZOOM : reset zoom window, clip limits in specified DAE.

Subfiles cannot be actively modified; there is nothing to modify except the item list. The user can find out which items are in the list by:

GIRDSF : go through the subfile and return the IDDADS of the items in it

An item can be modified in several ways:

GIALT : change set up information about the item (pickability, zoomability, absolute position)

GIDUP : duplicate an existing item and link it into the same or a different subfile. With GIALT this routine can reduce the cost of copying.

GIRECH : items can be totally restructured by reclaiming them into different subfiles or into a different order.

Two utilities apply to any IDDAD:

GIENST : returns the type of IDDAD. If IDDAD is an item additional information is returned

GIDELT : entity deletion. If IDDAD is a subfile, all items in the subfile are also deleted. The current version of INTERTEK doe not release item disk space to speed up the process. Rather, it "forgets" about the linkage and keeps the display file growing from the rear. Since the file is a scratch disk file, the approach is reasonable. The code is easy to modify if the alternate release-when-deleted strategy is desired.

2.6.4.4 Display File Processing

The user never sees the real display file processing routines in his code. The structure of these routines is hierarchical: information is formatted for one base working routine (GCWORK). When some sort of display file processing is needed a call is made to:

GCDRAW : figure out what is being processed and how the processing is to occur. GCDRAW handles:

1. A buffer of display commands already in core. GITEM and GIXTND use this feature to avoid reloading the array from a bead. This is particularly valuable in GIXTND because the first part of the item need not be redrawn.
2. Any individual IDDAD. The entity is always redrawn. That the parameter is an IDDAD rather than an array is flagged by a length parameter of 1. If the IDDAD is a subfile, the user must supply the DAE in which the subfile is to be drawn (used by GISHOW). Otherwise, the entity is reinterpreted.
3. If the LEN parameter is -1 or 0, the entire display file is redrawn.

The second phase (how full display file processing occurs) applies only to Step 3. If the LEN parameter is 0, the display file is redrawn. If it is -1, the display file is reinterpreted (but not redrawn) to check if a crosshair input is on a visible line.

As GCDRAW works on a DAE or an individual subfile, it first loads the clip limits, scale, and window center with:

GCDLIM : get limits from DAE into common GCDR. It then transfers control to:

GGDSF : interpret all items in a subfile.

Items, in turn, are processed by

GCDRIT : Interpret all display commands in an item.

Because items are really on disk, GCDRIT attempts to optimize its work by:

GCFILL : load array ISCR (common block GCGI) in increments of LSCR. This applies to an extended item--as many of the extensions as possible are loaded at one time.

All of these preliminaries set the stage for the worker of INTERTEK:

GCWORK : interpret the contents of the array passed in.

The interpretation is based on the display 'language' described in Figure 2.6.5. Much of the work is spent figuring out what type of commands are in the buffer. The two major components of GCWORK (the draw module and the pick interpreter) require further description.

The draw module is comprised of two pieces: a line drawer and a text output routine. The line drawer is fairly simple. An initial point is assumed. Each new coordinate is added. When the beam is off, the new location is merely bumped. Else, the two points define a line. The endpoints are scaled and translated to the DAE limits and then clipped for drawing purposes.

The text routine is a bit more complicated. First, text commands can stretch over a split buffer. In other words, ISCR may be filled before all text in a string has been output. This requires some care to retain the proper pointers when GCWORK is reentered with the rest of the string. The hardware text is clipped by calculating the rectangle which

the string occupies. A determination is made to see which characters are actually visible. Finally, zoomed text is a problem because the original character size is retained. The calculated envelope of the character string is much larger when the string is zoomed up, even though the string is the same size. Therefore, the string is centered in its theoretical envelope to retain some integrity.

The pick interpreter can ignore much of the clipping work. It must find out if the point input lies on a line or in a text string. A line must be visible for a strike to occur on it. A tolerance is assigned so a user does not have to position the crosshairs directly on a line. The software checks to see if the input points lies within the area of the line. (Figure 2.6.12). If so, a check is made to see if the distance from the point to the line is within tolerance through

$$d = \frac{Ax_0 + By_0 + C}{(A^2 + B^2)^{1/2}}$$

where $Ax + By + C$ is the equation of the line in question and (X_0, Y_0) is the point in question (see Figure 2.6.13). The picked point must merely lie within the rectangle of the text to be pickable (see Figure 2.6.14). GCWORK uses

GCC : see if point is clippable in this area
and COMPIO : low-level driver for asynchronous I/O to a Tektronix 4010 series direct view storage tube terminal.

2.6.4.5 Pick Processing

Pick processing is controlled by user definable categories. The categories are numbered from 0-63 and take on meanings for various types of picks. These categories are then used when determining if a pick is legal or not. Each pickable item must be assigned a category when the item is created by GITEM.

GICAT : assign an event type to a category. Legal types in INTERTEK are ignore, single, string, parameter, and button.

GCRCAT : retrieve the meanings assigned to a category

GCUCAT : unpack the categories to which this item belongs.

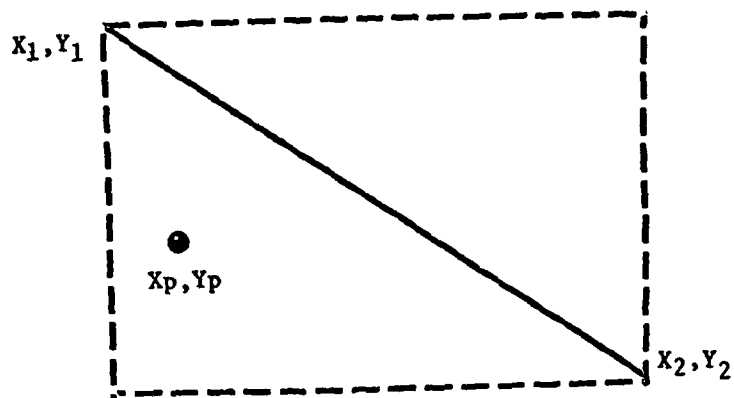


FIGURE 2.6.12. RECTANGLE CHECK

If X_p, Y_p lies within the rectangle, the test in Figure 2.6.13 is made.

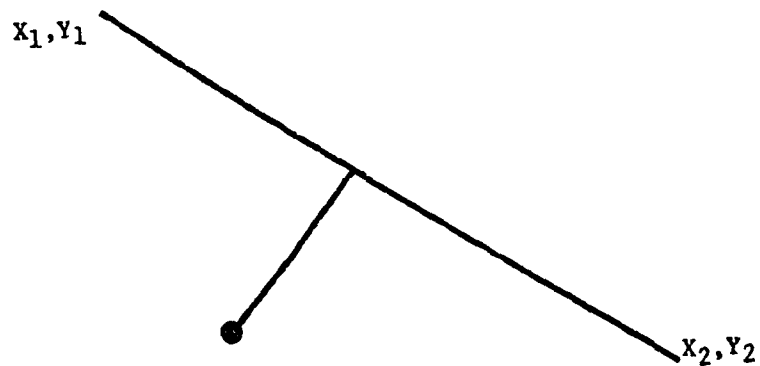


FIGURE 2.6.13. DISTANCE TEST

d is measured after the line equation through (X_1, Y_1) and (X_2, Y_2) is formed.

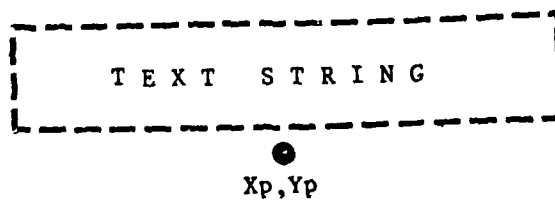


FIGURE 2.6.14. TEST PICK

X_p, Y_p must only lie within the text envelope.

In addition to items, keyboard keys can be assigned to a single category with

GIASID : Assign the category to the display code construct given. No ID block can be assigned. A key press pick can only be done if no display item was picked. Furthermore, the pick will only be honored if the category is a button pick type.

The actual pick processing is done by

GIBUTN : turn on the crosshairs and wait for a button pick. If the pick is not a button pick, an entry is made on the proper queue (single, string, parameter) for later retrieval. The actual processing of the pick is done by GCDRAW (0,1).

The other types of picks are retrieved from calls to:

GISNGL : format IEVENT, INF, ICAT, and ID for the pick on the single queue.

Entry points: GISTR: retrieve next string pick information from the string queue.

GIPARM: retrieve next parameter pick information from parameter queue.

Each queue is maintained separately. If the queues have not been emptied, the program can eliminate additional picks by

GICLRQ : eliminate single, string, and parameter queues.

Internally, much more work is done in processing picks than meets the eye. Much effort has been made to make the pick processing as rapid as possible. Two utilities aid in this process:

GCCHMM : see if the picked point lies within the rectangle occupied by an item.

GCGMM : grab the minimum and maximum limits (forming a rectangle) from either DAE or an item.

The picked point itself is retrieved by:

GCINPC : return X and Y of picked point and the key struck to generate the pick. X and Y are returned to 4096X4096 space.

All of this effort in pick processing is summarized by examining the techniques for actual determination of a legitimate pick as the display file is interpreted.

1. Get next DAE in list. If last DAE, pick not found.
2. Get clip limits. If picked point (PP) is not inside clip limits, go to 1.
3. Get next subfile in DAE. If no more subfiles, go to 1.
4. Get categories to which the items in this subfile belong (these categories are assigned to the subfile as items are linked in). If there are no categories or if all the categories are ignored, go to 3.
5. Get next item in subfile. If no more items, go to 3.
6. Get the categories for the item. If not in any category, or if all categories are ignored, go to 5.
7. Get lightpen sensitivity flag. If 0, item is not pickable, so go to 5.
8. See if PP is within the limits of the rectangular area within which the item lies (this information is collected as the item is being constructed.). If not in the area, go to 5.
9. Begin item interpretation with GCWORK using the scheme described in Section 3.4. If PP is not found go to 5; else, found so we can quit.

The strategy behind this algorithm is to eliminate large areas as quickly as possible. The display file can therefore be termed "area organized" because its component parts (down to individual lines and text strings) are looked at on an area-by-area basis before any attempt is made to calculate actual distance to a line. In complicated displays, this proves to be a distinct advantage over the test-it-all-because-its-there approach.

An adjunct to event processing is figuring out where the crosshairs were when an event was generated.

GIFETS : return the coordinates of the last location of the crosshairs.

An event need not be generated to use the crosshairs. Occasionally, it is necessary to put the crosshairs up and let the user

position them to return some location on the screen as repositioning information or the like.

GITRAK : turns the crosshairs on in a DAE for return by GIFETS. The crosshairs will reappear on the screen if the user positions the crosshairs outside the DAE.

2.6.4.6 Text Input

The technique for text input cannot be a true event in INTERTEK. The only mechanism is to use a read of some sort from the terminal. The programmer can use a standard FTN READ statement or

GIRDTX : Returns a text string and the number of characters input. GIRDTX reads from an internally connected file.

2.6.4.7 Utility and Miscellaneous Routines

Because the Tektronix is a DVST, the screen must be erased.

GIDFON : erase the screen and redraw the display file if so specified.

An audible tone will prompt (or wake up) a user:

GIALRM : issue a beep

Useful constants regarding terminal type are returned by

GCHIGH : return character height in 4096X4096 space.

Entry Points: GCWIDE: return character width

GCXMIN: return absolute XMIN for screen in
4096X4096 space

GCYMIN: return absolute YMIN

GCXMAX: return absolute XMAX

GCYMAX: return absolute YMAX

(XMIN, YMIN) and (XMAX, YMAX) are needed because the number of raster on the rectangular Tektronix scope differs between the 4010-4012 and 4014.

Errors are reported in INTERTEK on file OUTPUT.

GIERR : prints the error from the number provided.

GITRACE: prints a traceback of where an error occurred.

2.7 STAGING Model Graphics

2.7.1. Medium-Level Graphics Routines

A series of routines have been written to insulate the application from the perils of graphics programming using INTERTEK. This adds some execution overhead for repeated subroutine calls, but saves much re-coding and isolates graphics dependent code. Another set of low-level routines interface the INTERTEK package with the XY plotting package. The final set that is graphics-oriented takes care of scaling for INTERTEK based on zoom level.

2.7.1.1 Checks for INTERTEK

INTERTEK contains a buffer that may overflow. The buffer is the graphics information collection buffer (IBUF in DRAWBUF). When large display items are being constructed, this buffer is checked for overflow. The following subroutine performs this activity.

ICKBYT : Check if the IBUF will be overflowed. If so, generate an item if the first pass, otherwise extend the item. Normally only a call to ICKBYT need be made before a GU- call.

ICKBYT automatically generates an item and saves the IDAD if the IBUF buffer is overflowed. Every time a buffer is filled or flushed, a new item is generated or an old item extended by:

MITEM : Generates an item. The item will have the bead address of the owner if the owner is a bead address. It is placed in the category for that bead level (ICATS (IACTYP) in CATS. IACTYP is in common block PERMEN).

MEXTN : Extends an existing item.

2.7.1.2 Interface with XY Plotting Package

The XY-plotting package in STAGING is call-modeled on 4060 IGS. Several routines have been changed to interface the plotting package with

INTERTEK. These routines allow normal high-level calls and lets the programmer decide which pictures he wants in different items. This is accomplished by variable IDDADI in common block IGSIGS. Setting IDDADI to zero initiates a new item and a CALL METAZZ(0) terminates it. The program can then do what it wants with IDDADI before resetting it to zero to continue plotting. All features of IGS are pretty much standard except the default object space has been set to 4096X4069 rather than 4096X3072 to take advantage of all of the screen. The mode array Z(118) can be conveniently used through common block IGSCOM. All of the routines now call ICKBYT to insure that IBUF is not overflowed. XY Plot routines:

ERRZZ : Just a RETURN statement to save core.
LINZZ : A new routine which draws a line with beam on or beam off with GULIN.
METAZZ : Low-Level driver which formats all plot commands.
MODESG : Only calls RSETMG to initialize mode array.
RSETMG : Initiates mode array and scaling interface with the Display and Edit scalars by setting AMIN and AMAX in LIMIT to 4096X4096 and calling SCALST to set up the scale factors.
SETSMG : Recoded to eliminate unused mode sets thus saving much core.

2.7.1.3 Scaling and Coordinate Transformations

One of the more difficult tasks in INTERTEK is scaling because so much depends on the zoom level. Therefore, a series of utilities which communicate through common block LIMIT have been developed. The routines which initiate scaling are:

SET1 : Gets the display space for the model from a substructure
SCALST : Set up a 1-1(-1) display space with the largest dimension just fitting on the screen and the other dimension(s) adjusted so they are centered. The calculation is based on the zoom level (IZL in PIC) and the scale stored in SCALE and the offset in OFFSET in LIMIT.

Before a coordinate is displayed, it must be converted to the cartesian system and scaled into rasters. For completeness the converse for each routine is also provided.

CNVTOC : Convert the polar, cylindrical, or spherical coordinates in X, Y, Z in LIMIT to cartesian, replacing X, Y, Z .

UTORAS : Scale X, Y, Z in LIMIT user coordinates to rasters in IX, IY, IZ in LIMIT. If in 3D mode, to the projection before scaling by translating the point to the original, rotating to align the eye, and dividing by the perspective distance.

Entry UTORAT: Transfer the argument to X,Y,Z in LIMIT before scaling.

CNVTOU : Convert cartesian coordinates in X, Y, Z to user space.

RASTOU : Unscale rasters to user space. This converse works only for two-space because the inverse projection is not done.

Character sizes also depend on zoom level:

GETSIZ : Determine the size of a character depending on zoom level to be the smallest character visible. The calculation is returned in ITX1, ITX2, ITX3 in PIC.

As a model is drawn, the min and max coordinate values are traced to allow utilization of the largest possible picture. GETCOR does the actual trace while

UNSCAL : Sets current mins and maxes (CMIN and CMAX in LIMIT) to ridiculous values (CHIGH and CLOW in EDITT).

2.7.2 Construction of Model

The primary concern in constructing a display in this system is limiting the amount of information on the screen as quickly as possible. As the STAGING menu is traversed in this limiting mode, most of the routines set up mechanisms by which the user can select portions of the model to be displayed.

2.7.2.1 Initialization and Termination

The user must define what type of model he wishes to display. This can be two dimensions (2D), or three dimensions (3D). This distinction between 2D and 3D is straightforward. However, the software is sophisticated and permits any model to be displayed in either mode. This total flexibility allows the user to look at his model any way he wants to.

The following subroutines perform this activity:

INIT2D : Initialize 2D drawing mode.

INIT3N : Initialize 3D drawing mode.

The next step in limiting the problem is to select the data base level of interest. The following subroutines perform this activity:

STRACT : Set structures as the level to work on.

SSTACT : Set substructures mode.

ELEACT : Set elements mode.

NODACT : Set nodes mode.

These routines merely set IACTYP in PERMEN to 1, 2, 3, or 4. Termination of a picture is user-controlled: The screen must be erased by picking the ERASE SCREEN button. The following subroutine performs this activity:

ERASE : Resets the screen to its initial blank status (except for menus and the prompting message) by deleting the subfiles assigned to the display for each level (array IDSFM in common block PERMEN). The active mode and element beads are deactivated. If the mode switches from 2D to 3D or vice versa, a new display area entity (IDAEM in PERMEN) is generated. Finally, a guess is made for the initial scale by setting the scale to the boundaries of the model. (NOTE: The screen may be erased and the node or element beads not deactivated if an X-Y or contour plot will be generated. The erase process is controlled by variable IERASE in PERMEN).

2.7.2.2 Construction and Destruction of a Picture

The contents of any bead on any level of the data base can be displayed. Various mechanisms are available which activate the bead for display. Elements and nodes are drawn immediately, while substructures and structures require additional input. The base mechanisms for activation include:

- ALLON : Activate all beads on a level.
Entry point
ALLSON : Activate all nodes or elements in active substructure(s) only.
- ASERCH : Scan all beads for those that in active substructures only fall within the typed-in range.
Entry point
ASERCHA: Search all beads on a level for the attribute(s) falling within the typed-in range.
- RNGDRW : Draw all elements or nodes from the user's typed-in number range(s).
- ASUBS : Activate structures or substructures in response to a menu pick of names displayed.

Displays can be constructed from information already on the screen. These techniques allow the user to draw all elements attached to any one node or all nodes attached to any one element. The following subroutines perform this activity:

- LANODE : Draw all elements that own all the nodes on the screen.
- LFNODE : Draw all elements that own each node picked by the user.
- LFNOD : Make the call to IDRW for LANODE and LFNODE.
- NODAEI : Draw all nodes owned by all elements on the screen.
- NODFEL : Draw all nodes for each element picked by the user.

Drawing an attribute is a two-step procedure. The user must define which attribute value is to be displayed and then which pieces of the picture should have the attribute displayed. The following subroutines perform this activity:

- ACTATT : Save the attribute number from the attribute name

picked in variable IACATT in ACTSTR.

DRWATA : Draw the attribute for all beads on this level that are active.

DRWATT : Draw the attribute for each bead picked.

As well as drawing beads on a level, pieces of the display can be removed and deactivated. Deactivating a bead does not delete it from the data base. The analogous situation in which all but the picked items is also legal. Obviously, erase is the way to delete the picture en masse.

ALLOFF : Delete and deactivate all beads on a level.

DELPIC : Delete and deactivate all picked beads on a level.

SSAOFF : Deactivate all substructures or structures.

SSOFF : Deactivate substructures or structures from the list of active names.

RNGDEL : Delete and deactivate all beads corresponding to the element or node numbers typed-in.

RETAIN : Delete and deactivate all beads on a level except those that are picked.

RETAIS : Initialize the RETAIN process.

2.7.2.3 Drawing a Bead

The actual drawing process is performed by functions oriented about each level of the data base. A single routine, IDRW, will draw any bead supplied by branching to the proper drawing routine. The following subroutines perform drawing operations:

IDRW : The drawing controller. This routine will draw any arbitrary bead if it is not currently displayed (a non-zero IDDAD). If there is no picture, the proper drawing routine is called based on IACTYP in PERMEN. Once the picture is completed, the IBUF is flushed to generate or extend the item corresponding to the bead. A positive or zero value is returned by IDRW if no error has occurred in the draw.

- IDRWST : Draw a structure bead by drawing all substructures associated with it.
- IDRWSS : Draw a substructure bead by drawing all elements associated with it. This routine is ripe for modification because repeated lines are drawn. Each element is drawn independently but placed in the same item.
- IDRWEL : The work horse of drawing. The bead is checked for a proper number of down pointers. The routine then checks if this element is an extension to an existing IBUF (i.e. part of a structure or substructure). If the element is to be shrunk, its center is obtained (GETCEN). The actual display is generated from the element type irrespective of the number of nodes. This means that 'ROD' element can contain 50 nodes and still be drawn properly. More complicated elements are drawn using a connectivity array set up by routine ICON. The drawing algorithms is contained in the second column of array IELNAC in ELEMEN.
- ICON : Used by IDRWEL to set up a connectivity array for complex solid elements. Other element types can be added by expanding IELNAC and implementing the proper drawing algorithm.
- IDRWLN : Construct a line to (IX,IY,IZ) in LIMIT using the INTERTEK software with either beam on or off. The routine makes GULIN and the ICKBYT calls transparent to IDRWEL.
- IDRWND : Draw the node symbol specified. No checks needs to be made on the IBUF length because only two GU calls are present.

As well as drawing the physical model for any of the beads, specific values can be displayed for any of the attributes present. The following subroutines perform this function.

IDRWAT : Draw the bead (with routine IDRW) and add the attribute value specified in variable IACATT in ACTSTR. Each attribute is drawn centered in the bead (obviously, the center of a node is the X, Y, Z coordinate, and the attribute value is placed above the node symbol). By convention, if IACATT is zero, any drawn attribute is deleted and if IACATT is greater than LENATT(IACTYP) (the number of attributes in the list), the name is drawn. Some nodal attributes require graphical displays of arrows. These are determined from array IATDRW in IATTYP.

IGETCH : To ease the confusion of multiple attributes on the same picture, an abbreviation for the attribute name is displayed. Names and element types, are conventionally unlabeled. If a graphics display is to be made, IGETCH puts the start and end coordinates of the arrow in PMIN in PLANE.

ARROW : Pack into IBUF with GULIN, draws an arrow that begins and ends at the points specified. The direction (towards or away from 0,0,0) is specified by the third parameter. The arrow can be oriented arbitrarily in space because the IARSET description is rotated to align properly.

IARSET : Define the points and connectivity for a single or double headed arrow that is centered at 0,0,0 and points along the X-axis.

Once a display is constructed, the picture needs to be redrawn occasionally to change plot limits, shrink members, or reflect changes from editing. The following subroutines perform the redraw activity:

DRWACT : Redraw all beads on the level specified by IACTYP in PERMEN that are on the screen. The basic technique is to delete all active items for that level, and then redraw them. This routine is called by the Modify Picture routines when required.

Entry Points

DRWACN : Draws all active beads but does not erase them first. Used by deformed plot routines to superimpose a deformed plot.

DRWARR : Draws all beads from the array specified in the parameter list. Used by editing routines when a series of up pointers are queued as being affected by a change. For example, if a node position changes, all elements owning it are redrawn with DRWARR.

DRWONE : Draw the one bead in the parameter list used by editing routines when a change effects the display of that bead.

DRWNOC : Draw all active bead even if there is no display currently on the screen. Used initially by DYFILM to insure that the active structures and substructures are drawn.

DRWSS : Draw all active structure or substructures. Called from command tree 'DRAW PICTURE' button in structures or substructures mode.

2.7.3. Results Displays

Four different results displays are available. Deformed and dynamic plots show how X, Y, Z displacements affect the model. Attributes of various kinds can be molded into X-Y plots or contour plots.

2.7.3.1 Deformed and Dynamic Plots

The philosophy behind deformed plot mode is to set flags and let the standard model drawing routines do the dirty work. Most prominent of the flags is IDFLAG in DEFORM. If IDFLAG is greater than 0, a time dependent plot is generated through routine GETCOR. The time dependent plot looks at step n (ISTEP in DEFORM) and accumulates the displacement from steps 0 through n before plotting. If IDFLAG is less than 0, a mode shape or load condition at step n is generated which adds only the displacement at that one step. If IDFLAG = 0, and undeformed plot is in order.

Initialization and termination of deformed mode are handled by:

DYUNDE : Set undeformed mode (solid lines, IDFLAG=0).

Entry Point

DYDEF : Set deformed mode (dashed lines, IDFLAG=-1)

Deformed mode stays in effect until the user sets undeformed mode again.

When drawing the deformed plot, the IDDA0 of the last display drawn (deformed mode) stays active. The only way to remove the original undeformed plot is to ERASE the screen. The actual deformed plot is drawn from all active bead with:

DYDRDE : Draw deformed plot for all active beads.

The other deformed routines set up user type-ins.

DYLOAD : Put up the LOAD STEP type-in for deformed plots.

Entry points

DYMODE : Put up MODE SHAPE.

DYTIMS : Put up TIME STEP.

The processing routines allow the user to set:

DYSFPR : (entry to DYSFPU) Set variables SFACT or TIME in DEFORM based on a flag stored in NAMES(2) in EDITT.

DYSSTP : Set ISTEP in DEFORM.

Dynamic plots are somewhat more complicated than deformed plots. The user again sets up parameters before initiating the plot procedure. These parameters include:

DYSFPR (entry into DYSFPU): Set up SFACT or TIME in DEFORM.

DYSLIP (entry into DYSLIC): The user can extend the dynamic plot to include all steps between IST and IEND in EDITT, DYSLIP assigns those values.

DYSLIC : Put up range for dynamic plot steps.

When dynamic mode is completed, undeformed mode is reset.

2.7.3.2 X-Y Plots

X-Y plots are ultimately produced with routines on XYGRAPHPL which are call-modelled on IGSA060. These routines draw X-Y plots from arbitrary data sets. The routines in this section do little more than setting parameters in the mode array and collecting the data to be plotted.

As in other results displays, the user can select the piece of the data base for which the display will be generated by constructing the model on the screen. The following routines perform screen erase and initialization for the X-Y plot node:

INITXY : Begin X-Y plot mode by setting default plot parameters and erasing screen.

XYERAS : Erase the screen, but leave all active beads active by setting IERASE in PERMEN and calling ERASE. The parameter also sets a special activate-but-do-not-draw when the user activates more data for plotting.

Entry points

XYRET : When in the X-Y plot module, the user can erase the plot he has just constructed. To retain consistency, all active nodes and elements are deactivated but the activate-no-draw mode stays in effect.

XYRET1 : Leaving X-Y plot mode erase the screen, deactivates active nodes and elements, and returns to activate and draw mode for nodes and elements.

The user actually activates data with the same tree and the same routines as he does in constructing the model. He must then select which attributes he wants drawn on the X and Y axes. When he picks the attribute, the following routines are used:

GETDAT : Collects the data from the attribute selected for the proper axis from all active beads of that type. If it is the X-axis, the values are sorted into increasing order.

PROCXA : Selects the X-axis for data collection.

Entry point

PROCYA : Selects the Y-axis for data collection.

TABRDR : Redraw one line on graph.

REPLY : Replots an X-Y plot to reflect new plotting options selected later.

RSCXY : Rescales the X-Y plot for the absolute min and max for all curves displayed and redraws it.

SORTXY : Sort the X- or Y- data in ascending or descending order.

When working with tabular data, a different set of routines is responsible for activating data. The major processing routines (GETDAT and PROCXA) have modifications in them to retrieve from a single table rather than geometry-oriented data.

TABACT : Determines which tables are plottable
(from IDBFTY)

Entry points

TABACX : put up table names for choice as x-axis

TABAXY : put up table names for choice as y-axis

TABPRC : Process table pick and put up type-in box for user selection of pertinent data to plot.

TABDM : Process type-in for part of table to plot.

TABVAL : Lets user retrieve values from table

TABIND : Treats pertinent array as subscript values. Useful when plotting a singly dimensional array.

Entry point

TABVAL : Use the table as values.

The user has control over a large number of features about graph style as well as its contents. These choices include grid style (full or tick marks), titles (X and Y axes and graph), scaling mode (linear/linear, linear/log, log/linear, log/log) and plot style (solid, short or long dashed lines, point plot, or connected point plot). These parameters are set initially and can be reset through:

INTFLG : Set up initial plot style to be linear/linear, tick marks, XTITLE, YTITLE, GRAPH TITLE, solid line plot.

Entry Points

GSFUL : Set full grid.

GSGXGY : Set linear/linear.

GSGXLY : Set linear/log.

GSLXGY : Set log/linear.

GSLXLY : Set log/log.

GSTIC : Set tick marks.

LSLDH : Set long dash plot.

LSPLN : Set point and line plot.

LSPNT : Set point only plot.

LSSDH : Set short dash plot.

LSOLD : Set solid line plot.

PROCCHR: Set new plot character from type-in.

PROCTIT: Set new X and Y axis and graph titles from type-in.

QTITLE : Display old X and Y axis and graph titles

PROCTIT: Set new X and Y axis and graph titles from type-in.

The user can set new limits for his plot that will be constant until changed. The term AUTO means that Display and Edit will automatically scale the data as it is retrieved.

TABXYP : Process type-in of new XMIN, YMIN, and XMAX, YMAX

The sort order (in SORTXY) can also be changed by

TABXI : Sort X increasing

Entry Points

TABYI : Sort Y increasing (default)

TABXD : Sort X decreasing
TABYD : Sort Y decreasing
TABYN : Do not sort X
TABYN : Do not sort Y (default)

Individual lines in the plot can be changed as well:

TABPEN : Activate lines on graph for pick
TABLIN : Work on the line picked

The line style of the curve can be changed by the entry points of
INTFLG or with:

PROCCHR: Change plot character
Every n^{th} point can be plotted:
PROFAC : Set new repeat factor
The picked line can be deleted:
TABDEL : Delete line and squeeze other down.

The scale/offset of each line can be changed. This is useful if
looking at tables with data in different units (i.e Volts/millivolts).

TABSCL : Change scale/offset

2.7.3.3 Contour Plots

The contour plotter in the STAGING system is limited to
contouring an attribute value at points in the $Z=0$ plane defined at nodes
or element centroids. Currently a maximum of 128 points may be contoured.
The following routines do the work:

CNTINT-- Initialize contouring default values in common block
CNTDE.
GETCNT-- Fetch the attribute data from the data base from the
activated nodes and elements. The data is stored in
common block CNTDE. Integer data is converted to
floating point. Mins and Maxs are calculated.
CNTPRD-- Put up current contour parameters for user modification
via type-in.
CNTPRE-- Edit and accept typed-in parameters.

CNTOR -- Main contouring subroutine which calls the routines below.

CONHUL-- Determine convex hull of points in plane to be contoured.

TMESH2-- Create triangular mesh over the convex hull.

TMESH3-- Iteratively improve triangulation by using other quadrilateral diagonal if it increases the minimum angle between triangle edges.

TRIORD-- Perform a reordering of the line indices so that attribute value at initial point is less than or equal to attribute value at terminal point.

SCAN -- Trace through the triangular grid to extend contour lines for each requested value.

The user has control over number of contour levels, the values of the contour levels, contour plot labeling, and using scale factors on the contour value labels.

2.7.4 Modifying the Picture

Various mechanics are available for modification of a constructed picture. This section will discuss these functions in the order in which they occur in the command tree.

MORDRW : General redraw of active beads.

2.7.4.1 Split Screen

The user can generate up to a four-way split screen. The number of splits active at one time is kept in NUMSPL in PERMEN. When SPLIT SCREEN is picked, the working DAE is moved to occupy a smaller area. The following routines perform this activity:

MOSPLT : Decrease the zoom of IDAEM (if necessary) to fit the next split on the screen. The routine creates a new DAE for the split information. Then erase the screen.

If NUMSPL = 4, the request is ignored.

SETWZ : Reset the virtual window and picture limits to existing DAE with GIZOOM.

The user can generate a free or frozen copy of the working DAE (IDAEM). A free left side shows the existing subfiles in the blank DAE. This means that any modification to the display in IDAEM (except local modification) will be reflected in the new DAE. A frozen left side copies all shown items, making them impossible to change. The following routines support this activity:

MOFRZ : Show all subfiles in the new DAE (IDAEM (NUMSPL) in PERMEN).

MOFRE : Copy all items in IDAEM to IDAEM (NUMSPL).

The following routine deactivates the split screen mode:

MOSOS : Delete all DAE's and subfiles except the working DAE (IDAEM IN PERMEN) and subfiles (IDSFM (1-4) in PERMEN).
Then erase the screen.

Erasing the screen only erases the working area. The remainder of the image is left intact in the display file.

2.7.4.2 Shrink Members

A shrink capability allows each element to be drawn at 80% of its normal size. This generates a slot in which adjacent element boundaries will not overlap in either 2D or 3D. The following routines perform this activity:

MOSSHK : Set shrink mode (SHRINK = .8 in LIMIT)

Entry Point

MOUSHK : Termination from this segment. Sets SHRINK back 1.0.

MOASHK : Shrink all members on the screen. If a substructure is drawn, all elements in the substructure will be shrunk.

MOISHK : Shrink only the members picked.

2.7.4.3 Change Plot Limits

The user can enter new plot limits in three coordinates systems. The picture will then be rescaled to the new limits. For entering the new limits from the keyboard:

MOREC : Sets rectangular entry mode.

MOCYL : Sets polar or cylindrical entry mode.

MOSPH : Sets spherical coordinate mode.

The user can also fill the space up from a constructed model with:

MOFILL : Rescale the picture to the current mins and maxes (CMIN and CMAX in LIMIT) seen since the last screen erase. This means that CMIN and CMAX are not readjusted if items have been deleted from the screen without erasing the entire screen.

The actual work is done by:

MODRW : Redraw the picture if the limits have really been changed.

2.7.4.4 Restore Original Picture

The following subroutine is used to restore the original picture:

MOZSRS : Restore original picture by (1) restoring the working DAE to full screen and its initial zoom and virtual window and (2) unshrinking all shrunk members. The current perspective and rotation is kept.

2.7.4.5 Perspective View

This feature applies only to 3D mode. The user can change the ten word view control array (VU and VU3D) by typing in new values for eye position, point looked at, and projection plane.

SVIEW : Process the typed-values. A null type-in restores to the current view, thus eliminating any rotation.

SETVU : Calculate the new view. Calculate a perspective or orthogonal rotation menu.

Table 2.7.1 and Figure 2.7.1 illustrate the meaning of the various viewing parameters. The user can display a 3D model in 2D mode. In this manner, the user can change with lightpen in the X-Y, X-Z, or Y-Z planes. The new projections are done with:

FRONTVU: Set X-Y plane

SIDEFU : Set Y-Z plane

TOPVU : Set X-Z plane

2.7.4.6 3D Axes

In 3D mode, axes centered at (0,0,0) of the display and oriented along the user's X, Y, Z coordinate system can be drawn.

MOAXES : Draw the axes with ARROW and label them with MOLAX. The item IDDDAD is stored in IERASE in PERMEN. (IERASE is a (-,0,+) switch exclusively; an IDDDAD is always +). The axes are arbitrarily placed in IDSFM(1). Picks to MOAXES act as an on/off switch. If the axes are there, they are deleted, and vice versa.

MOLAX : Label 3D axes with the characters X, Y, and Z.

2.7.4.7 Rotate

In 3D mode, the user can type in new values for angles. They are processed by:

SROTET : Rotate image from host.

TABLE 2.7.1. THE VIEW CONTROL ARRAY

NAME	MEANING	PERMISSIBLE VALUES
		(Any Numeric Type-in is in the User's Coordinates)
X LOOK AT	X, Y, Z Coordinates of the point you wish to look at on the model. Only meaningful when a perspective Projection	an X, Y, Z in the model. If the character C[enter] is entered, the center of the model is calculated as the point looked at.
Y LOOK AT		
Z LOOK AT		
X EYE	X, Y, Z Coordinates of your eye. If none specified at infinity, a perspective projection is made.	An X, Y, Z outside the volume occupied by the model. An entry of +1 or 1 yields in an orthogonal $+\infty$ and -1, $-\infty$. C[enter] means the same as for point looked at.
Y EYE		
Z EYE		
PROJ PLANE	The projection plane for the closer the projection plane to the point looked at, the larger the picture.	A distance from the point looked at. If 0, a distance halfway between the point looked at and the eye is calculated.
FRONT CUT	A distance from the point looked at that controls the cutoff planes. The front cut is the distance along +Z, the back cut along -Z. If both are zero, the const int parameter takes over. A depth cut between FRONT and BACK is generated otherwise constant intensity from parameter CONST INT is used.	Any positive value.
BACK CUT		
CONST INT	otherwise constant intensity from parameter CONST INT is used.	
CONST INT	Constant intensity	1-16 (not implemented in Tektronix version of STAGING)

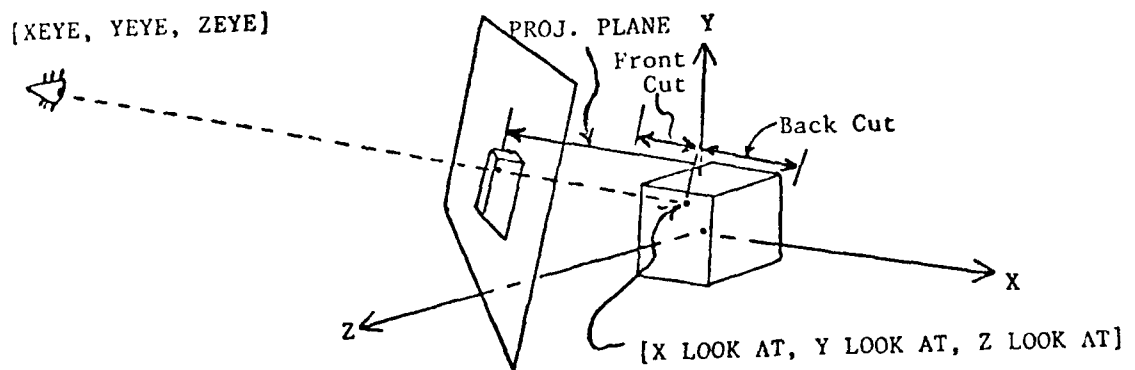


FIGURE 2.7.1. 3D VIEW CONTROL

The current rotation angles are displayed by

SROTST : Put up current rotation angles and ask for increments.

2.7.4.8 Zoom

A general zooming capability is provided by the following routines:

MOBOTH : Reset zoom and virtual window for IDAEM to its initial
(value of IZL, IWY, IWY in common block PIC.

MOZPLS : Zoom one level more (enlarge).

Entry

MOZMIN: Zoom one level less (compact).

MOZCRS : Zoom around crosshairs.

MORCNT : Recenter display file around crosshairs.

2.7.5 Editing the Model

The editing portion of STAGING is general in nature for the user controlled variables in the data base: attributes, down pointers, and up pointers. Complications occur in the code because changing one piece of the data base may effect several different images on the screen. Most notably, changing a nodal coordinate will affect the display of all elements that own it. Deleting parts of the data base has the same tendency to change the display of its owners. Creating new beads on any level will not effect the display until linked into the data base with up and/or down pointers. All editing routines use data base routines to incorporate the changes.

2.7.5.1 Initialization and Utility

The key to editing is the same as for displaying. The user must activate specific beads on a unique level to edit. The following routines are called when 'CHANGE FEATURES' is entered.

EDINIT : Sets no beads active for editing (all beads of this level on the screen) if this is a change in data base levels (i.e. NODES to ELEMENTS) or the first time editing has been entered for the level. Otherwise, the currently active beads for editing remain active until 'ACTIVATE FOR EDITING' is picked.

ECHNIT : Eliminate the active for editing list when 'ACTIVATE FOR EDITING' is picked.

The active for editing list is kept as two lists (IACTL and IACTH) in CREATR. These lists, both of length LACT, hold the left and right ends of a list that is scanned (IACTL(I) to IACTH(I) for I=1 to IACTPT) during the actual editing phase. The user can construct these limits through:

ESLOT : The base routines for activation. The routine fills IACTL and IACTH until LACT is overflowed.

EACTPN : Activate a bead from a pick on a displayed item.
Entry Point

EACTP : Activate a bead from a displayed structure or substructure name.

EALLS : Activate all beads on the screen.
Entry Points

EALLD : Activate all beads for this data-base level

EALLSS : Activate all beads in the active substructures.

ETYPSV : Activate all node or element numbers typed in.

When a change is made to a bead that affects its owners, the edit module will redraw all those owners which are displayed to provide immediate visual feedback of the change. The up pointers are saved in groups of 20 to minimize the number of times a bead is redrawn by:

EQUEUP : Saves the up pointers for a bead. Each up pointer can occur only once in the list. A parameter of zero flushes and redraws the up pointer list.

Because different modes are available for activating beads for

editing, the following higher level traverser is used when more than one bead must be changed:

IENEXT : Returns the next bead in the string by managing the IACTL and IACTH lists. The links are followed based on ICFLAG in CREATR, which is set during activation for editing.

Entry point

IENIT : Initializes IENEXT

2.7.5.2 Changing Fields in a Bead

The user has access to the attribute list and up and down pointers in any bead. In the edit phase, he can change attributes for more than one bead. He is constrained to work on only one bead when changing up and down pointers.

2.7.5.2.1 Changing Attributes

Two modes of operation are allowed. The multiple bead mode (where more than one is active for editing) allows the user to change any or all attributes to a new value. The single bead mode allows the user to change any field (including the name) from a list of current values in the bead. The same routine handles both modes:

EATS : Start the display of attributes for changing by calling EATF.

Entry point

EATCH : Does the actual work of changing attributes. If a currently displayed attribute is changed, the bead is redrawn. If a node point is changed, all displayed elements that own the node and the node itself are redrawn. Changing an element type also triggers a redraw.

A special editing mode is available for 2D displays. If one node

is active for editing or newly created, a new (X,Y) value can be assigned by moving the crosshairs:

MXYI : Initiate tracking.
MXYP : Process the move by getting the coordinates of the tracking symbol, converting them to user coordinates, and redrawing affected nodes and elements.
TRKOFF : Turn tracking off (crosshairs).
Entry Point
TRKON : Turn tracking on.
EMOVE : Put up centroid of node, element, or substructure.
Entry Point
EMOVEP : Process move.
EUPPNT : Do work updating pointer(s) in node, element, or substructure.

2.7.5.2.2 Adding and Deleting Up and Down Pointers

Modifications to up and down pointers are limited to one bead at a time. This constraint has been chosen to simplify user interaction. The main mechanism for the user to specify up and down pointers is by use of the crosshairs. He may select any bead currently displayed on the screen or choose from a list of numbers or names. As a last alternative he may also type in values. Subroutines DBUPT and DBDNT do the majority of the work. All routines take care of cross-linkage automatically. For example, if a down pointer is removed from a bead, the bead is removed from the up list of the owned bead. The following subroutines are used:

ECHOND : Initiate down pointer mode.
Entry Point
ECHONU : Initiate up pointer mode
EADOWN : Add down pointers from a lightpen pick of a displayed item. The picture is redrawn for the bead after all picks are processed.
Entry Point
EAUP : Same as EADOWN, but for up pointers.

ERDOWD : Set parameters for ERD to remove a down pointer from a name or number displayed. Used on structure and element levels.

Entry Points

ERDD : Same as ERDOWN except remove is for a displayed item. Used on structure, substructure, and element levels.

ERDOWU : Same as ERDOWN except remove is for up pointers. Used on substructure, element, and node levels.

ERDU : Same as ERDOWN except remove is for a displayed item's up pointers. Used on substructures, elements, and node levels.

ERD : Remove up or down pointer. Checks are made to insure that the pointer is legitimate before the removal is attempted.

ENDOWN : Add down pointers by picking the name of down pointer. Used for adding to structures.

Entry Point

ENUP : Add up pointer by picking the name of the up pointer. Used for adding to substructures and elements.

ERNOLD : Delete down pointers from a structure or element from a list of the down pointers.

Entry Points

ERNADD : Add down pointers to structures from the list of the available down pointers.

ERNADU : Add up pointers to substructures or elements from the list of available up pointers.

Adding elements to substructures provides the user more mechanisms than the set listed above. Included in the repertoire of substructuring techniques are collecting elements with certain attributes or within a specified geometric region through:

ESERCH : Look for elements that have particular attributes and add them to the active substructure.

PLINIT : Set up initial volume to search which is bounded by the limits of the current display.

PONPEN : In 2D mode, the user can position a rectangle with crosshairs over the desired region.
Entry Point

PCOLL : Reads up the coordinates of the rectangle after positioning.

PTYPE : In 3D space, the user can type-in the coordinates of the volume which are processed by this routine.

PORG : Do actual search and add operation for elements totally within the volume bounded by PMIN and PMAX in PLANE.

Quite often a user has made an error in the node list of elements. In this case, he may substitute one node for another by:

ELREPL : Process type-in for nodal replacement.

2.7.5.2.3 Creating New Beads

A user can create new entities on any level and define attributes and up and down pointers with the tools in the previous section. The actual processing and creation operations are done with:

ECREAN : Initiating name type-in for creation.

ECPROC : Process name type-in. A new slot is created with this name if there is no occurrence.

IECPRC : Does check to see if typed in name is in the data base.

Substructures and structures can be merged to provide a newly created bead on either level containing the down pointers from both. The technique looks at the down pointers of the other substructures with:

EMERGE : Add down pointers from each structure or substructure picked to the newly created bead.

2.7.5.2.4 Deleting from the Data Base

When a bead is deleted, all beads on the level above it are affected. Therefore, up pointers are queued with EQUEUP as well as the actual deletion operation in:

EQPEN : Delete the bead corresponding to the displayed items.

Entry Point

EQPENN : Delete the bead corresponding to the
displayed name (substructures and structures).

EQNUM : Delete the bead corresponding to the displayed number
(element or node) typed in.

2.7.6 Graphical Input

The user is allowed to pick items on the screen other than the menu generated from the command three. In general, these are variables dependent on the data base and displayed in pictorial form as a wire frame model or as textual information in the form of attribute names or bead name.

2.7.6.1 BASICS

The picking mechanics allow the program to selectively turn the "pickable" attribute on and off for different parts of the model. Each level of the data base displayed belongs to a different category (ICAT in CATS). Each category may have a different pick meaning (MEAN in CATS). Only pickable displayed text resides in CATS(5) and MEAN(5). These arrays allow different parts of the display to be pickable at any one time. The pickability is initiated by setting a member of CATS to an event type other than ignore with:

SETCAT : Set the category for the data base level (1-4) or text (5) to the event type specified.

2.7.6.1.2 Picking Displayed Data Base Beads

The usual means of activating the crosshairs on allows the items on a level to be string pick sensitive with:

ONPEN : Make all items on this level (IACTYP in PERMEN) pickable. If none are currently displayed, issue an error message.

Entry Points

ONPENA : Turn on all beads as single picks.

ONPENE : Turn on elements as string picks (for activating nodes by elements).

ONPENN : Turn on node as string pick (for activating

elements by nodes).

ONPENP : Unused.

ONPENS : Turn on all beads as string pick.

NODON : Check if nodes are pickable before ONPENN is called.

EONPEN : Turn on crosshairs in editing.

EONPND : Activate crosshairs for next level down in the data base.

The processing routine then works on the picked items until the string is exhausted (ID(1) in IDENTs is zero).

2.7.6.3 Picking Text Items

Picking text items requires construction of the list of names before picking can occur. Utilities are available for putting various types of text on the screen.

The same problem occurs here as it does in text-editing. There may be too much information to display at once. In this case, an auxilliary entry point is provided for the processing routine to call when the last ID of the set is processed. The second entry point checks to see if there is more to display and puts it up if needed. A flag is returned to the processor indicating that there may be more picks from the new information. The IER flag in ERROR is then set to override the command tree action so the command tree stays where it is rather than taking its normal action.

The two basic routines put up bead names or attribute names:

TYPNAM : Display the names of the beads in NAMES (IST-IEND) in EDITT in the menu area as string picks. The ID block of each item corresponds to the bead address.

Entry Point

TYPVAL : Display the alphanumeric string in NAMES.

SELATT : Display the attribute names with a NAME or NUMBER descriptor and a DELETE ATTRIBUTE choice. The ID for each attribute is its position in the attribute list.

The items are set up as single picks.

Entry Points

SELATC : Continue list is LENATT(IACTYP) is greater than MAXSCR until list is exhausted.

SELAT1 : Used in X-Y and contour plots to display all of the above except the DELETE ATTRIBUTE choice when choosing data for an axis.

2.8 Material Property Data Base (MPDB)

2.8.1 Introduction

The MPDB is split into two levels - a master bead and a number of specific material beads. It is in Data Handler format. The file information and the format of the beads are described below.

2.8.2 MPDB Modification Procedure

The STAGING material property data base (MPDB), accessed by the user in the pre-processor module, can be reconstructed, changed, or modified using the following cataloged procedures. These procedures are available on file PROFIL, ID = STAGING3. See Appendix A for procedure listings.

Create

Procedure Name: MPDBNEW

Input: - None

Output: - PFN = MPDB, ID = STAGING (automatic catalog)

The created file is a basic skeleton MPDB and does not contain any materials. New materials are then added using procedure MPDBADD.

Add or Revise:

Procedure Name: MPDBADD

Input: LFN = TAPE1 - formatted input of material (see below).

lfn=MATER,pfn=MPDB,id=STAGING (automatic ATTACH and
EXTEND)

Output: Modified MPDB

Print file output

Input Instructions for Tape 1

Adding to or revising the STAGING Material Property Data Base (MPDB) consists of preparing a formatted input file (lfn = TAPE1) and running the procedure MPDBADD. The following describes the card formats of TAPE1.

Control Card - (I5)

NOR - Number of Requests (Addition + Revisions)

The following group of cards must be supplied for each of requests.

Material Identification - (2A1-, I5, F10.5)

GENI - Generic Name of Material, e.g. STEEL, ALUMINUM

SPCI - Specific Type of Material, e.g. SS307, 6061,T6

NPT - Number of different temperatures for which properties are defined (maximum 10).

RO - Mass Density of Material

Material Property (4F10.5) - One Card for each temperature

TEMP - Temperature

E - Young's Modulus

- Poisson's Ratio

- Coef of Thermal Expansion

2.8.3 MPDB Format

File Name = MATER

Logical file number = 5

Buffer = IBK in common block MATDB

Buffer length = 160

Block size = 2 PRU's

Bead Format =

1. MASTER BEAD - 21 words

2. SPECIFIC MATERIAL BEADS - 45 words

MASTER BEAD

Word 1 - No. of Generic Names - Integer

Word 2 -

to 10 - Generic Names - Alpha

Word 11 - Unused

Word 12

to - Address to specific Material - 0

Word 21 - Bead

SPECIFIC MATERIAL BEAD

Word 1 - Specific Name - Alpha-numeric
word 2 - Bead Address for Next Specific Material Bead
Word 3 - Unused
Word 4 - Material Density
Word 5 - No. of Temp. Points
Word 6 - Temperature (T_1) - Real
Word 7 - Young's Modulus at T_1 -
Word 8 - Poisson's Ratio at T_1 -
Word 9 - Coff. of Thermal Expansion at T_1
Word 10 - Temp. (T_2)

Word 45 - Coeff. of thermal exp at T_{10}

2.9 STAGING Code

2.9.1 Overview of STAGING Code

Two concepts were used to make the STAGING system as versatile as possible. First, a top-down strategy was employed. Secondly, many concessions were made to make the code as easy to add to (or delete from) as possible.

The top-down strategy dictates that routines be as modular as possible. STAGING has a proliferation of routines that serve one purpose only. In this way, much code does not have to be rewritten to change the way it functions. This will simplify future modification greatly. Generally, only low-level routines will need to be changed if the current technique does not appear adequate. When similar strategies are employed, routines are grouped with entry points and switch settings. The entry points are named similarly to the main subroutine name to help locate them. Again, this should make modification of low-level algorithms much easier.

Code modification is a traditional problem in working with large systems. Constraints abound that are hardwired to a specific constant. For example, a buffer may be defined to be 100 words long and every time that length is used the programmer has written '100'. When the buffer needs to be increased or decreased in size, all those references to '100' must be changed. This problem can cause large amounts of time to be wasted because there is invariably one more reference that was not found.

To avoid this problem, STAGING uses many variables to indicate lengths of important pieces. The only hardwired constants in the code refer to the data-base levels of 1=structure, 2=substructure, 3=element, 4=node and 5=tables. This concept of geometric grouping through the data base has proved to be most flexible for the problems thus far encountered. The five levels are the most fixed concept in the entire system and the code looking at the levels uses constants throughout, especially when differentiating between the low levels of elements and nodes and the high levels of structures and substructures. Other instances of 'hard-wired'

code may exist, but the general rule is that the code uses variables throughout. All of these variable are set up in common blocks (with UPDATE COMDECK's) and initialized in one of the block data routines or assigned in the initialization routine for that module.

Variable names are as mnemonically descriptive as possible. To assist in figuring out the meanings, comments are provided with each 'COMDECK' listing the meaning of each variable. Names generally follow the FORTRAN naming conventions. When a variable needs an 'INTEGER' specification, it is contained in the COMDECK for that variable. New routines will never need to make a COMMON variable INTEGER or REAL.

File manipulation is handled through FORTRAN callable subroutines. All local files that need to be attached are returned before use to prevent system errors caused by illegal permanent file utilization. The systems programmer needs to be aware of the following local files:

- OUTPUT - Location of errors from NSRDC Data Handler and conversion routines. Can be connected or disconnected before execution. The file is not returned or rewound before use.
- TAPEO - The permanent file for the user's data base.
- DESKRCH- To prevent system failure from destroying TAPEO, the user really works on local file DESKRCH, which is a literal copy of TAPEO.
- DANDE - File for menus for DISPLAY and EDIT.
- GLOBAL - File for menus for GLOBAL commands.
- EXEC - File for menus for EXECUTIVE.
- PREP - File for menus for PREPROCESSOR commands.
- POST - File for menus for POSTPROCESSOR commands.
- MATER - Material Property Data Base

The code in all routines is FORTRAN with the exception of a few specialized routines for character manipulation and permanent file utilization written in COMPASS.

The code is written to utilize the CDC segmentation loader. This implies that only constants are initialized by DATA statements. When a constant could change, the variable must reside in a COMMON block that is

1

'saved' in the segmentation environment. The FORTRAN trick of putting a DATA statement in to initialize a switch does not work in the segmentation environment without the COMMON save feature.

A concession was made for core savings because the FTN compiler uses quite a large amount of space for argument lists. Because of this, argument lists are short and used only when needed. Many values are passed through COMMON blocks. Functions are also used extensively, especially when a single value is to be returned.

The code uses the INTERTEK and NSRDC Data Handler packages extensively. In order to allow code to be transported to other devices, as much centralization of routines which use these packages has been done as is practical.

One of the features of STAGING is the ability to recover from unexpected errors. These errors can be internal in the program (such as a mode error), or caused by the user (user abort). The same general technique is used to recover all errors.

The error recovery package uses system PP routine RPV to guarantee that STAGING cannot be aborted easily. The recovery procedure is initiated by a call to routine MARK, which is part of the Battelle error recovery package. MARK flags the location where the recovery package will return when an execution termination condition is detected.

Once recovery is initiated, the fateful question 'DO YOU WISH TO CONTINUE (TYPE Y OR N)?' is asked. If the answer is 'N' or 'NO', the system returns to INTERCOM command mode.

Typing anything else resumes execution. The internal STAGING recovery process is straightforward. The general cause of an abort will be during construction of a display. The user can abort a long running display construction manually by typing %A. The reaction is the same in all cases.

The menus are picked up at the same spot where they left off. The ERASE routine sets up the proper subfiles and DAE's to allow another picture to be constructed. The ERASE mode is kept constant as beads will not be deactivated if the X-Y or contour plot modules are being executed.

As an aid to the system programmer, the options of DMP and DMPX are available at recovery even though they are not advertized to the user. Three listings are handy at this point. First, a listing of STAGING with the FTN R=1 cross-reference map is adequate. This allows rapid access to specific variables relative to the start of any one subroutine. Second, a segload with a partial map lists the start of all subroutines in the application. Third, a listing of the appropriate command tree is mandatory of find out what routine was called for a particular button.

Logic errors do occur. They can be easily traced, however, with the proper listings. The most effective technique is to abort the program. This invokes the recovery package and core can be examined at leisure with the DMP option. The Alphanumeric dump feature is imperative for finding out what segment is in core at any one time. The R=1 cross-reference map will then give the proper pointer to the variable(s) in question.

Mode errors can also occur. These automatically invoke the recovery package. Mode 1 errors are generally caused by incorrect segmentation. The problem areas must be traced through the recovery package DMP option. The usual reaction is one of shock to find a particular segment in core. Mode 2 errors generally result from undefined variables in the data base. Using the DMPX feature of the recovery package, the UNDEFINED flag in STAGING can be looked for in an X register. The unique value is 4000765432107654321B.

If necessary, intermediate prints can be placed into the proper routine. Entry point PUTLIN in routine GETPUT can be used to display debugging messages on the terminal screen. OUTPUT is generally disconnected and contains system errors from FTN and Data Handler.

The most helpful hint to be given for STAGING is in the construction of a brand-new data base. It is somewhat counter-productive to rely on the conversion routines and then work with the data base because it is often desirable to work on the data base as it is constructed. To that end, it is much more convenient to create a "dummy"

data base, and then use the data base handler routines for the rest.

The dummy data base need only consist of two nodes and one element. The structure and substructure are defined automatically. This also allows a convenient mechanism for initial definition of the attribute list.

For example, the following routines can initialize, add to, and terminate a data base.

```
SUBROUTINE DBIN
  DIMENSION IATN(3).ATT(3).node(2)
c   set up element attribute as type
    IAT = 1
c   set up node attributes as X-Y-Z coordinates
    IATN(1) = 1
    IATN(2) = 2
    IATN(3) = 3
    CALL CNINIT(IATN,3,IAT,1,0,0,0,0)
c   define dummy nodes
    NODE(1) = 9999999
    ATT(1) = ATT(2) = ATT(3) = 0
    CALL CNNODE (NODE,ATT,3)
    NODE(2) = 9999998
    CALL CNNODE(NODE(2),ATT,3)
c   and dummy element
    ATT = 3HROD
    CALL CNELEM(9999999,NODE,2,ATT,1)
c   end the process and re-open the data handler field
    CALL CNTERM
    CALL DBINIT(5HTAPE0)
    RETURN
  END
```

Code to add a new element or node is similar. An example of a node add follows. It is assumed that the nodes start at one and are incremented for each new node. X is an array of length 3 containing the new X-Y-Z location.

```

SUBROUTINE ADNODE(X)
  DIMENSION X(3), IATLOC(3)
  *CALL  DATBAS
  DATA NODNUM/0/
C
C   for first node, initialization must be done
C
      IF (NODNUM .NE. 0) GO TO 10
      IATLOC(1) = ISACT(6HX-CORD,4)
      IATLOC(2) = ISALT(6HY-CORD,4)
      IATLOC(3) = ISALT(6HZ-CORD,4)
C
C   find the previous pointer in the list
C
      IPREV = IDBLFT(IHEAD(4))
C
C   get the new bead
C
10   NODNUM = NODNUM + 1
      IPREV = IDBADB(NODNUM,IPREV)
C
C   insert X,Y,Z and make sure limits of substructure are maintained
C
      DO 20 I = 1,3
      CALL DBCHA(IPREV,IATLOC(I),X(I))
20   CALL ECHKL(I,X(I))
      RETURN
      END
      The element add must put in up and down pointers. The pair
      CALL DBDNT(IPREV,NODBED)
      CALL DBUPT(NODBED,IPREV)

```

will cross reference the nodes and elements, where IPREV is the current element and NODBED is the bead corresponding to the node number. The

substructure cross-reference is done by

```
CALL DBUPT(IPREV,IHEAD(2))
```

```
CALL DNDNT(IHEAD(2),IPREV)
```

where IHEAD(2) is obtained from COMDECK DATBAS and IPREV is the current element.

To terminate, one must merely get rid of the dummy beads and set the LKUP array for later use.

2.9.2 Program Library Maintenance

All update to STAGING source code is done using cataloged procedures described in Section 2.1.

2.9.3 Segmentation Strategy

The STAGING system is dependent on the CDC segmentation loader. This strategy was chosen because it offers far greater flexibility than the traditional overlay mechanisms. For this flexibility we pay the price of difficulty in segmenting a very large system into a restricted core size without impairing run-time efficiency. Many errors can be caused by certain combinations of segmenting methods and coding methods. A subprogram may work correctly only if never swapped out of memory. No local variable values are saved but DATA statements reload initial values. Good programming practice will ensure that the routine functions correctly even if swapped out between calls. This allows the segmentation directive designer to work independently of the code. He needs to know only the tree of who calls who and some rough idea of frequency and order of calls. As a matter of practice, all routines should be included in the segmentation directives. The defaults for segments for undeclared subroutines are often insufficient and frequently illegal. Particular care must be given to ensuring that common block contents are swapped out to disk so that the contents are saved when brought back into memory. Care is needed to ensure that common blocks are in core when routines needing them are executing. (The segmentation loader will not automatically reload the necessary common blocks).

The STAGING segmentation scheme has six segment levels:

- 1: Root
- 2: Swappable Common Blocks
- 3: Application Subroutines
- 4: Miscellaneous Subroutines
- 5: Graphics Subroutines

6: System and Data Handler Subroutines

The integrity of the segmentation scheme depends on structuring the directives so that no routine calls another routine which is closer to the root. Thus routines in level 3 may call routines in level 5 but may not call routines in level 1. Similar restrictions occur within the trees in the individual levels.

The root level contains the DRIVER(STAGING main program), the error recovery main subroutine (RCOVER), the menu tables, and many common blocks. Level 2 is necessary because of the large size of the common blocks used only in the contouring package. Level 3 contains most of the routines called directly from the menus. It is divided into separate trees corresponding to the major modules of STAGING. Level 4 contains menu management routines as well as routines called from more than one major module. Level 5 contains INTERTEK and various other graphics routines. Level 6 contains system routines (Record Manager and FTN library), Data Handler routines, and various ubiquitous general utility routines.

The STAGING segloader directives for the six levels are listed in Appendix B.