NASA CR-
141596

AEROPHYSICS RESEARCH CORPORATION
TECHNICAL NOTE

JTN-10
VOLUME I

THE DLG PROCESSOR - A DATA MANAGEMENT EXECUTIVE FOR THE ENGINEERING DESIGN INTEGRATION (EDIN) SYSTEM

VOLUME I - ENGINEERING DESCRIPTION AND UTILIZATION MANUAL

By: C. R. Glatt and W. N. Colquitt

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Johnson Spacecraft Center
Houston, Texas 77058

December 1974

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-		

4. Title and Subtitle	5. Report Date
THE DLG PROCESSOR - A DATA MANAGEMENT EXECUTIVE FOR THE ENGINEERING DESIGN INTEGRATION (EDIN) SYSTEM VOLUME I - ENGINEERING DESCRIPTION AND UTILIZATION MANUAL	December 1974
	6. Performing Organization Code
7. Author(s) C. R. Glatt and W. N. Colquitt	8. Performing Organization Report No. JTN-10 Volume I
	10. Work Unit No.
9. Performing Organization Name and Address Aerophysics Research Corporation Houston, Texas 77058	11. Contract or Grant No.
	13. Type of Report and Period Covered CPFF June 1973, Dec '74
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Johnson Space Center Houston, Texas 77058	14. Sponsoring Agency Code EX 4

15. Supplementary Notes

Final Report

16. Abstract The DLG processor is a Univac 1100 series computer program designed to read, modify, manipulate, and replace symbolic images. DLG is controlled by a set of user supplied directives and operates from a data base of stratified information which can be merged with the symbolic images. Data bases can be constructed and maintained in the mass storage media using the DLG directive language.

17. Key Words (Suggested by Author(s))
EDIN, language, data base, storage retrieval, data management

18. Distribution Statement
Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	31	

*For sale by the National Technical Information Service, Springfield, Virginia 22151

AEROPHYSICS RESEARCH CORPORATION
TECHNICAL NOTE

JTN-10
VOLUME I

THE DLG PROCESSOR - A DATA MANAGEMENT EXECUTIVE FOR THE ENGINEERING DESIGN INTEGRATION (EDIN) SYSTEM

VOLUME I - ENGINEERING DESCRIPTION AND UTILIZATION MANUAL

By: C. R. Glatt and W. N. Colquitt

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Johnson Spacecraft Center
Houston, Texas 77058

December 1974

PREFACE

This report describes a computer program called the DLG Processor - A Data Management Executive for the Engineering Design Integration (EDIN) System. The program was written in support of NASA Contract NAS9-13584, "Extended Optimal Design Integration (Extended ODIN) Computer Program." The study was conducted during the period from June 1973 through December 1974, with funds provided by the National Aeronautics and Space Administration, Johnson Spacecraft Center, Engineering Analysis Division. The contract was monitored by Mr. Robert W. Abel. The report is presented in two volumes:

Volume I - Engineering Description and Utilization Manual

Volume II - Programmers' Manual

The report specifically describes a user-developed data processor which is integrated with the Univac 1100 executive system and is interfaced to the EDIN data base.

TABLE OF CONTENTS

THE DLG PROCESSOR

A DATA MANAGEMENT EXECUTIVE FOR THE ENGINEERING DESIGN INTEGRATION SYSTEM

VOLUME I - ENGINEERING DESCRIPTION AND UTILIZATION MANUAL

By: C. R. Glatt and W. N. Colquitt

SUMMARY

The DLG Processor is an 1100 series Exec 8 computer program designed to read, modify, manipulate and replace symbolic images. DLG is controlled by a set of user supplied directives (or language elements) which provide the basic capabilities of the DLG Processor as follows:

1. Language elements for the construction and maintenance of a data base which is independent of any other computer program.
2. Language elements for processing information generated by other computer programs.
3. Language elements for automatically retrieving data base information as input to any other computer program.
4. A simple technique for editing and interrogating the data base and for generating summary reports of data base information.

The DLG processor can be used to form a linkage between engineering technology modules through the manipulation of common information in the data base. The use of the system for this purpose requires the prior assimilation of the following basic components:

1. A library of independent technology programs including the DLG processor.
2. The control card sequences for the execution of the technology modules.
3. The setup data for the technology modules which perform the desired analysis function.

The basic components are often available without additional program development. The program sequencing and intercommunication required to integrate the basic components into a design simulation are established using the Exec 8 run stream concept. (See Reference 1) Data base requests for common information are established by the formation of skeletonized data elements containing execution control cards and/or technology module input data. The skeleton elements containing DLG directives are processed or filled out by the DLG Processor using data base information. The result of the preprocessing is a partial run stream which is acceptable to Exec 8 for performing computerized design tasks.

INTRODUCTION

The EDIN system provides a balance of data management techniques which consider the inherent capabilities of the computer operating system, past efforts in the storage and retrieval of stratified data and the recent development of some flexible paging techniques for the transfer of information between the computer core and the mass storage of the computer. The Univac Exec 8 system provides the resources for the storage of large complex data files, for the storage and retrieval of the files and for the cataloguing protection and backup of the files. The executive system has several processors with instruction sets for manipulating the data retained in mass storage. A limitation on the operating system capabilities arises in accessing the subfile level of information in the system files once the file is addressed.

The EDIN data management system is designed to subdivide the files in a manner that will allow the data which is retained in mass storage to be accessed at any level from the single parameter level to a large matrix of data. Rather than constructing an extensive single computer program that attempts to be everything to everyone, the EDIN data management system provides a three-level data management capability. This approach permits the individual designer using the system to make his own decisions with regard to the storage method and techniques. It also permits the flexibility of using existing data sources not specifically created for EDIN.

The three levels of the EDIN data management system are built upon one another as illustrated in figure 1. The lowest level deals with the interface between the data in mass storage and the computer operating system. The file level of the data management system is provided by the Exec 8 software and consists of the file utility processor FURPUR, the file administration processor SECURE and other system level processors. The system processors are accessed using Exec 8 control statements. Therefore, file level software may be used directly by the designer for transmitting large structured blocks of data or the files themselves to be accessed by the programmer who seeks economy above all else. The file level constitutes the foundation for all higher level data management components.

The second level of the EDIN data management system provides the mechanism whereby the files can be organized into blocks of data called pages. Pages of information can be organized in a number of ways and names can be given to each page. A pointer system or directory is maintained by a Fortran callable software

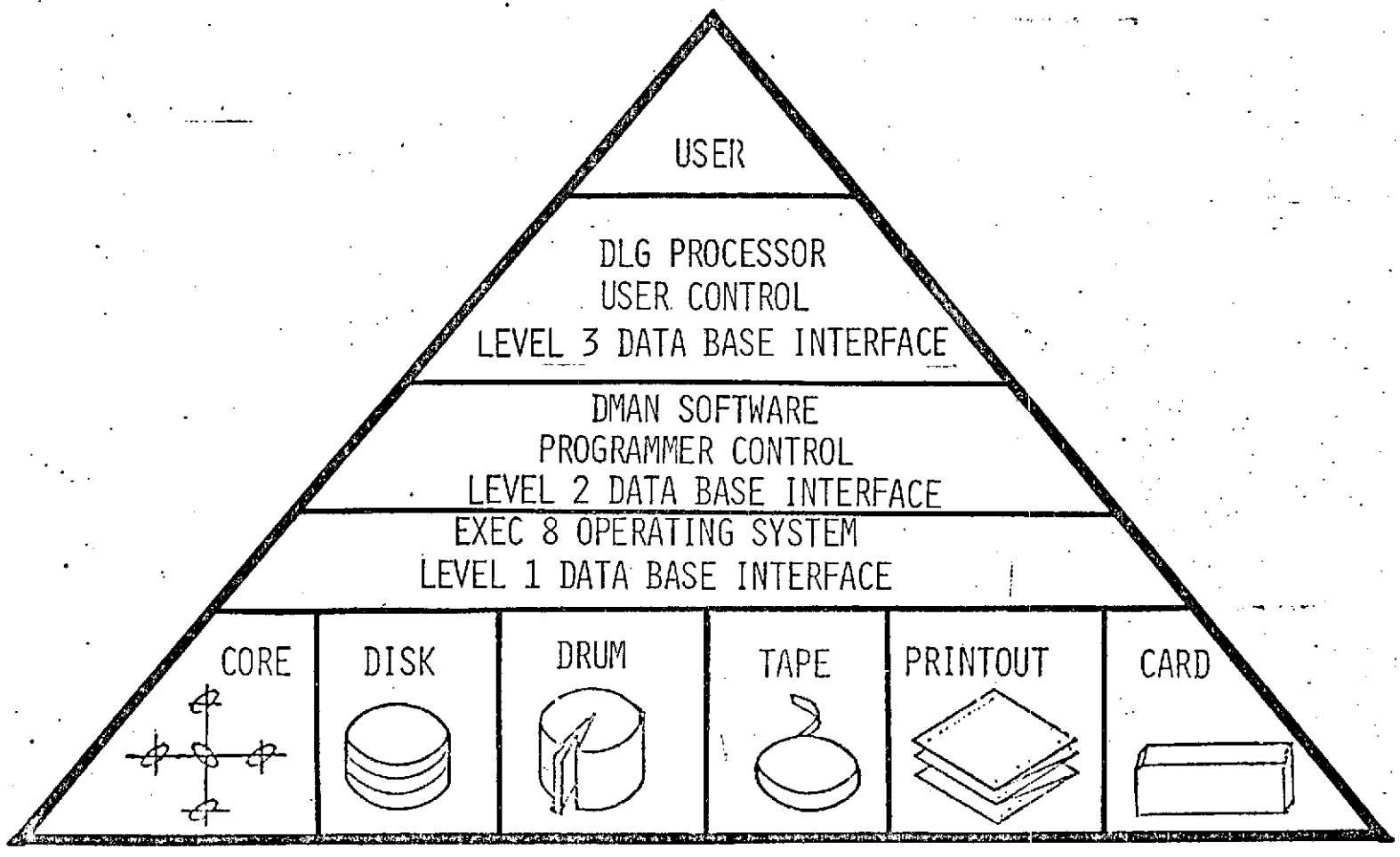

FIGURE 1 EDIN DATA MANAGEMENT SYSTEM.

package, called DMAN, a subroutine utility package maintained in the EDIN library.

The third and highest level of the data management system is provided to make the system more usable to the designer who may not be a programmer. The capability is provided in the DLG processor which is designed to maintain a data base of stratified information, the stratified data can be selectively accessed and merged with the input stream of the EDIN technology programs. This level also provides the interactive language structure which allows the designer to sit at a remote terminal and interact with the data base directly as he develops a design. The DLG processor also contains routines for processing the output from the technology programs for the storage of design information in the data base.

Although the user may access the data base through any of the three levels, it is the lowest level maintained by the Exec 8 system which actually stores and retrieves the data. Exec 8 handles all of the underlying data management functions including file assignments, file directories and maintenance and security procedures as well as the data block transfer to and from mass storage. The Exec 8 system is discussed in the previous section and a thorough treatment of the first level data management is provided by Univac in the appropriate User Documentation. The following discussion deals with the second and third level of the EDIN data management system.

Complete instructions for use of the DLG language elements in conjunction with the basic EDIN components are presented herein. In addition, an interface technique is described for allowing any program in the library to update the data base. The technique does not influence the stand-alone operation of the program.

ENGINEERING DESCRIPTION

The DLG procedure for implementing the linkage of the technology modules is to read output data from one module, insert a selected subset of the resulting data into a stratified data base and then selectively extract this and other stored data for placement into the input stream for other applications programs. The linkage is illustrated in figure 2. The effect is to provide a unified analysis involving several modules operating from a single source of data. Repetition of execution sequences can be triggered by looping criteria residing in the data base.

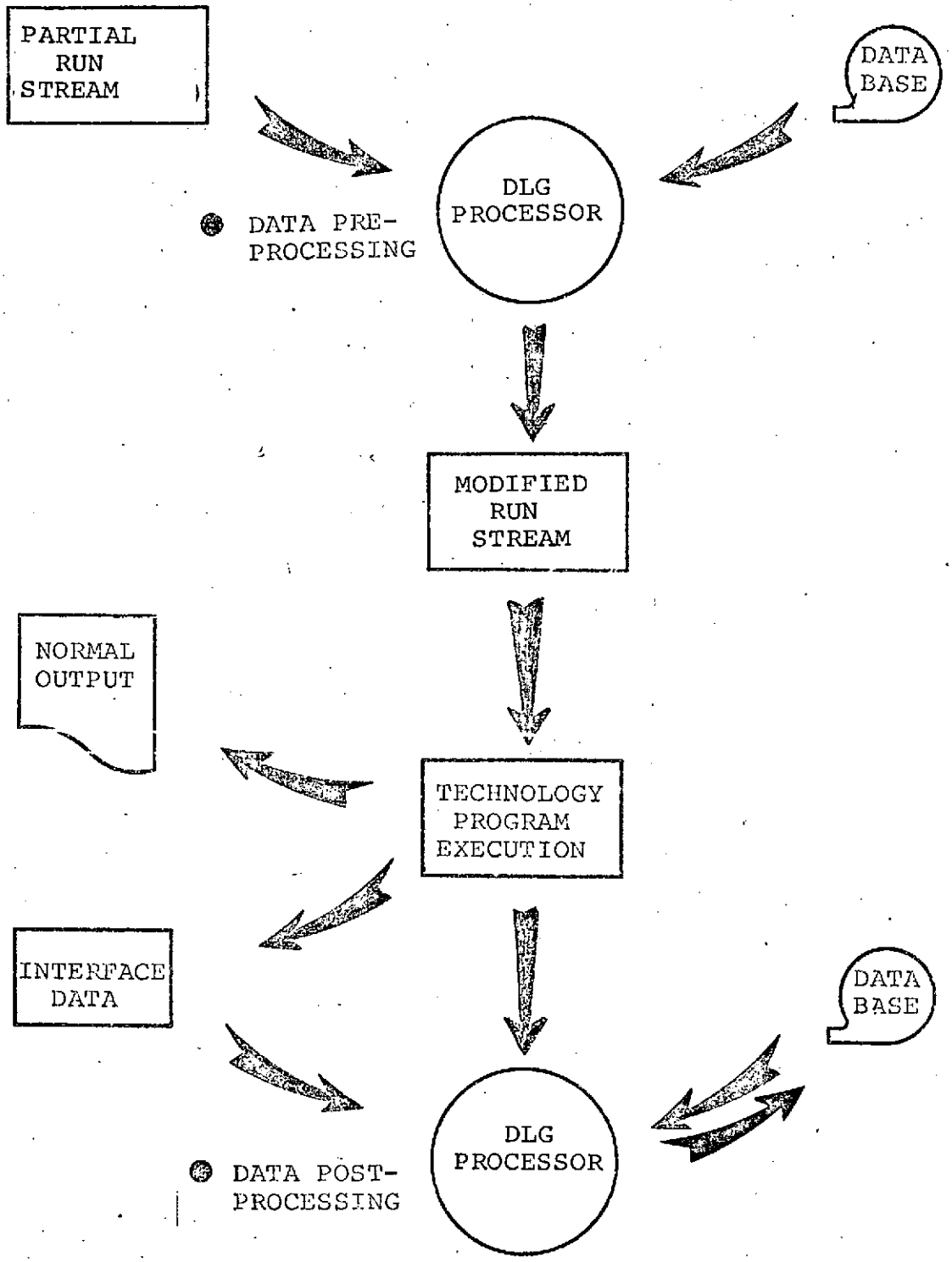

FIGURE 2 DLG PROCESSOR FUNCTIONS.

Concepts and Definitions

The following concepts and definitions which may be new to the reader will be helpful in understanding this document:

Processor	An absolute program element which is executed with a special Exec 8 processor control statement: `@name eltl,elt2` and which is interfaced with the elements named on the processor control statements.
Data Base	File of information which is subdivided into named pages of data accessible by the DLG processor. Each page is further subdivided into named parameters and arrays.
Technology Module (Application Program)	An independent computer program which will receive or generate data base information.
Interrogation	The process of retrieving information from the data base. The disposition of the retrieved data is dependent upon the directive employed.
Directive (Also Command)	A language element used to specify a DLG Processor's action or function.
Data Management	A class of DLG functions which control and manipulate data base information. These functions include the creation of data base pages, the adding and defining information in the data base, printing and many others.
Data Storage	A special class of data management functions which are designed specifically to store data generated by a technology module.
Run Stream	A sequence of data images which constitute a computer run.
Partial Run Stream	A portion of a run stream which can be merged at any point in the run stream through an @ADD control statement.

Language Structure and Usage

The language for controlling the DLG Processor consists of control directives which are summarized below:

'name'	Replace name with information from the data base.
'ADD	Replace specified information in the data base.
'CHANGE	Change values in common IDLOG.
'COMMENT or '.	User description with null effect.
'CREATE	Greate a new data base.
'CSF or 'ER	Submit executive control statement.
'DBLIST	Print the names of all random access data bases on the data base file.
'DEFINE	Place description in data base directory.
'FORMAT	Format free data base information in place.
'INSERT	Insert binary SDF data in place.
'ON	Mode activation.
'OFF	Mode suppression.
'PRINT	Print data base information.
'USE	Specify a circular data base search.
'UPDATE	Update a specified data base.

Each language element is delimited by a pair of apostrophes (') to distinguish DLG input from the technology module input data and execution control cards in the run stream. For example, in the language element,

'NAME.....'

NAME is the name of the command or directive (i.e. CREATE or ADD). The remaining information within the delimited region is ancillary to the specific directive. The DLG Processor scans the partial run stream for delimited information and performs the function specified therein.

The control directives provide the means by which data base storage and retrieval functions can be performed. They command the transfer of information from the analyst to the data base, from the data base to the technology programs and from the technology programs to the data base. In the transfer of information from the analyst to the data base, the ADD, DEFINE and INSERT commands are used. A comment command is also available for annotating data. In the transfer of information from the data base to the applications program, the replacement command is used. Data from the data base is transferred by name to the input stream of the technology programs. In the transfer of information from the technology programs to the data base, a special extension of the ADD command is employed. An interface file of ADD-like commands can be generated by any technology program for later processing by the DLG Processor.

Construction and Maintenance of Data Bases

Data base construction and maintenace are accomplished by the use of the three control directives, CREATE, UPDATE and USE. The CREATE directive creates a new data base which is stored on a temporary disk file by the same name. By permanently storing the data on the disk file, the user may access the information at a later time. The use of the UPDATE directive in conjunction with previously created data bases, allows the user to modify the data contained therein. The USE directive specifies a search sequence of existing data bases.

The basic construction of data base pages is similar but varies in total size and specific characteristics which can be specified by the user at creation time. Data bases consist of two distinct parts, a free storage array of packed information where the actual data is stored and a directory of names and pointers to the data in the free storage array. The directory and the data base have certain attributes which are assigned by the CREATE directive.

```
'CREATE name, attribute = value, attribute = value'
 (DATA)
```

Any or all of the attributes may be specified by the user as follows. If not specified, the default values will be used.

Attributes	Default	Description
LTOTAL	1024	Total number of computer words allocated to the specified data base.
NWORD	2	Number of computer words per data base entry.
DIRLEN	47	Total number of directory entries allocated to the data base.
LENDES	5	Number of words of descriptive information associated with each entry in the data base.

Each entry in the directory is the name of a data element or the name of an array of data elements. Up to DIRLEN entries may be made. Each entry may have a short description (LENDES-2) stored with it. Each data element occupies NWORD computer words. In establishing the size of the data base (LTOTAL), the user must consider the total number of elements of data desired in the data base, the directory length, the desired length of description and the number of words per data element. If NELMT is the total number of elements, then:

```
LTOTAL = DIRLEN*(LENDES + 3) + NELMT*NWORD
```

After a data base is initially established, the UPDATE directive may be used for adding or modifying the information contained therein. The form of the directive is:

```
'UPDATE name'
(DATA)
```

Attribute parameters which are established once by the CREATE directive may not be specified by the UPDATE directive. If the use of the existing data base pages without modification is desired, the directive:

```
'USE name,name,...'
```

is used. No attributes may be specified.

The basic control directive for entering information is the ADD command. It permits a variable name and value or values to be placed in the data base:

```
'ADD name=value,value,'
```

If the name is a new entry, any number of values may be added. The number of values associated with the name when it is first added is also the number of locations reserved in the data base for that information. Later modifications to the information can not create more data base space. The values may be real, integer, hollerith or logical. A single ADD command may be used for creating or updating many information sets.

```
'ADD V1 = 25., V2 = 30, V3 = ALPHA, V4 = .TRUE.,
     A = 10., 15., 20., 25., I = 4, 5, 6
    V4 = 5 * 0.,'
```

The data type is defined by the input data type. Upon retrieval, the data type is determined by the characteristics of the stored data. The format of the ADD statement is patterned after the FORTRAN NAMELIST feature and indeed has many of the same characteristics and usage rules. For example, all name/value sets are separated by commas (,); all elemental values of an array are separated by commas; the entire statement (command) is delimited. In the case of NAMELIST, the delimiter is a dollar ($) sign; in the case of ADD commands, the delimiter is (') . The rules for entering hollerith information and multiple constants differ from NAMELIST.

The ADD command has additional capability not present in NAMELIST. The value associated with the ADD name may be a previously defined data base variable name:

```
ADD V1 = V2,
```

The effect of the above command is to transfer the information associated with V2 to the data base space assigned to V1. V1 may or may not exist prior to the ADD command. If V1 did not exist, space will be created in the data base as the information is transferred. If V1 did exist, then the information in V1 will be replaced by the information in V2. The transfer of information from one data base location to another is generally limited to scalar quantities. The ADD command may also be used for combining existing data base information with other data information or constants:

```
'ADD V1 = V2 * V3,'
```

The operation illustrated above indicates a multiplication of the two numbers on the right side of the equal (=) sign prior to

transferring the resulting information to the space allocated to V1. Any algebraic operator may be employed as follows:

+ addition
- subtraction
* multiplication
/ division
** exponentiation

More than one operation may be performed on the right side of the equal (=) sign.

'ADD V1 = V2 + V3 * K,'

Up to ten operations may be performed within a single ADD command. The operations start from the equal sign and progress to the right. The first variable is combined with the second. The result of that operation is combined with the third. The result of that operation is combined with the fourth, etc. It should be noted that the hierarchy or order of the operations does not conform to FORTRAN arithmetic. For example, in the above illustration, V3 is added to V2, then the sum is multiplied by K.

If the data base were so defined, the directory may contain space for storing descriptive information. Depending on the value of CREATE parameters, LENDES space is reserved for each name entered equivalent to:

LENDES -2

The default value of 5 provides three computer words (30 characters) for descriptive information. The DEFINE command is used to store the descriptive information in the directory:

'DEFINE name, description,'

DEFINE name = value, description,'

Name is a new or existing data base entry. If the name exists, the description is added. If not, the name and description are added. If the name is a new entry, then the value may be used to reserve space in the data base for data elements to be added later. The absences of value on a new entry results in the reservation of space for a single data element.

Construction of Partial Run Streams

There are three major considerations in the construction of design partial run streams:

1. Establish a design data base and control card data base for use in the simulation.

2. Construct the desired design sequence.

3. Provide data base interfaces within the input data sets for the technology programs.

Data bases are established at the beginning of the run or transferred from mass storage on an one-time basis using the techniques discussed above. The design data base is defined first by use of a CREATE, UPDATE or USE directive.

Execution of a Sequence of Technology Programs. - Now consider the problem of sequential execution of one or more technology programs using the EDIN system.

DLG preprocesses the data associated with each program and constructs a modified run stream. The preprocessing function will be discussed later. The physical linkage between the control card sequences of the various applications programs and DLG is an Univac operation system utility program, @ADD. @ADD is executed to link the technology program sequences just preprocessed by the DLG program. The linkage is continued by Exec 8 until an @FIN control statement is encountered. Details of the operation of the executive system are contained in reference 1.

Conditional Branching. - The DLG processor can direct the transfer of control forward or backward in the simulation stream by the use of dynamic run stream modifications. This capability is achieved by the following control card directive language elements:

```
@SETC 'V2'
@TEST OP/'V1'
@JUMP label
```

The @JUMP control statement establishes a label name in the execution sequence where control may be skipped to. The @SETC control statement establishes a data base value in a special system register which is tested by the @TEST control statement to determine if the @JUMP control statement should be processed.

The combination of the DLG Processor functions and the Exec 8 dynamic run stream modification capabilities permit a complicated system of analysis loops to be constructed for satisfying a variety of matching constraints.

The @TEST employed provides a standard set of tests on the variable set by the @SETC control statement:

@TEST TG/Value	for (V1 > V2)
@TEST TLE/Value	for (V1 $\leq$ V2)
@TEST TE/Value	for (V1 = V2)
@TEST TNE/Value	for (V1 $\neq$ V2)

As noted previously V1 and V2 are constants or variables constructed in the design data base through use of the ADD command or generated by any technology program in the synthesis and passed to the design data base.

Retrieval of Design Information

The retrieval of design information from the data base is accomplished by two basic control directives:

1. Replacement Directives.
2. INSERT Directives.

The control directives are strategically placed into the technology program input data sets. The augmented data sets form skeletonized input for the technology programs. Each data set is headed by UPDATE or USE control directives corresponding to the appropriate data base pages.

Upon execution of the DLG Processor, the data sets are scanned for control directives delimited by apostrophe pairs (') for identification. Each encounter with delimited information causes DLG to perform the indicated replacement or insertion action. The result of all encounters within the skeletonized input for each technology program is a modified partial run stream which can be merged with the run stream through the @ADD control statement.

Replacement Command. - Data base information may be entered into the technology program data sets by means of delimited data base variable names entered in the precise location where the requested data is to be placed. The delimiters define the field width for simple replacement (i.e. a single data element). The

DLG processor will replace the variable name and delimiters by its data base value and rewrite the card image in the modified run stream. Input formats such as NAMELIST-like inputs and rigidly formatted input can be accommodated by the procedure. For example, in a true NAMELIST input, a data base variable would be entered as follows:

```
NAME1 = 'V1 ',
```

NAM1 is the name of the NAMELIST variable. V1 is the data base name. The delimiters specify the field width to be employed in replacing the data base name with the corresponding data base value. Similarly for a formatted input where data normally appears on a card within a specified range of columns, the data base replacement procedure is simply:

```
'V1   '
```

The delimiters are placed at the appropriate card columns defining the field for the data element. The data base name is placed within the defined field adjacent to the first delimiter.

Additional capability is available when namelist input is used by the technology program. Entire arrays may be transferred to the input stream using the following procedure:

```
NAME = 'VARRAY'
```

where VARRAY is a data base array name. If the data in VARRAY contains more data than can fit on one card record, additional 'cards' are created and placed in the modified input stream to accommodate the excess data.

Data base variables and constants may be combined much like the capability described for the ADD command above. For example, the operation:

```
'V1 * V2'
```

illustrates how the product of V1 and V2 may be used as the replacement value of data base variable. The product never resides in the data base but only in the modified input stream. V1 must be a data base variable name but V2 may be a data base variable or constant. More than one arithmetic operation may be performed within the delimiters:

```
'V1 + V2 * V3'
```

Up to ten operations may be performed within a single command. The hierarchy of operations is the same as that described for the ADD command.

In general, array elements may also be used in the replacement command:

```
'V1(n)'
```

where n is a data base variable, or

```
'V1(5) * V2(6)'
```

One exception from this capability is in dealing with the first element of an array:

```
'V(1)'
```

The above command specifies the replacement of an entire data base array and is therefore not an acceptable command for replacing the first element of an array. The dilemma may be avoided by use of the following two cards:

```
'ADD NEW = V1(1)'
'NEW'
```

The ADD command defines a new data base variable which will contain the element V1(1). The replacement command will place the variable NEW (i.e. V1(1)) in the modified input stream. In general, all ADD command capability described under data base construction is applicable when placed in the skeletonized element of a partial run stream. The 'ADD command instructions are performed but the 'card' which contains the command is not transferred to the modified input stream.

A special feature of the replacement command is the element-by-element combination of entire arrays or the combination of arrays with constants. As an illustration, consider the example:

```
'V1 * V2'
```

where the data base variables V1 and V2 are arrays. The above command specifies the element-by-element multiplication of the arrays. If one array has fewer elements than the other, the combining of elements ceases after the shorter array is exhausted and the rest of the longer array remains unchanged. The variable V2 may be a constant or data base name. Multiple operations may be performed using the hierarchical rules described above.

Insert Command. - Unlike the replacement command which extracts information from the design data base, the INSERT command transfers information sequentially from a formatted data card file. Starting and stopping positions within the file may be specified as follows:

'INSERT name1 = n/m, name2 = j/k'

The effect of the command is to search the named system files, name1 and name2, for integer card numbers n through m and j through k. If the card numbers are omitted, the entire named file is transferred to the modified input stream. The end-of-file may be specified for the named file by replacing m or k with the character string, EOF. The card containing the INSERT command is not transferred to the modified input stream.

Comment Command. - In addition to previously described commands, there exists a special "command" for identifying data. The format is:

'. comment'

No action is performed as a result of this command. It is useful only as an identifier for other data. For example, consider a technology program which uses formatted input (i.e. numbers with no identifiers or names associated with them). The comment command may be used to identify the data elements within the input data stream. The effect of processing this command through DLG is simply the replacement of the command with blanks. If the resulting card image is entirely blank, then the card is not transferred to the modified input stream.

Report Generation

A special feature of the ODINEX program is its ability to produce user generated report formats augmented with data base information. The report generator is applied in the following manner.

The report data is formatted by the user to provide any descriptive information desired. The report data may contain data base information through the use of the communication commands described above. One example of card image in the report might read:

WEIGHT OF THE SYSTEM IS 'WGT' POUNDS

In the above illustration, WGT is a data base variable name. The DLG Processor replaces the data base name and delimiter 'WGT' with the information stored in the data base. The report is printed through the normal computer output channels. The insertion of a report in the simulation stream does not effect the normal sequence of events for the simulation. The report may contain carriage control characters in column 1 of the report data cards.

1 - Eject a page before printing.

0 - Skip a line before print

Any number of reports may be generated during a simulation. The format of the individual reports is tailored to the needs of the particular study. Once the format is established, it can become a permanent part of the simulation stream. Any of the features of the DLG language, including scaling and adding data base information, are used in a completely free field report format.

Technology Module Interface Package

The communication of information from a technology program to the EDIN data base generally requires modification of the applications program. This modification is usually trivial and requires little programming knowledge to accomplish. The objective of the modification is to create a special file of information which contains a format suitable for reading by the DLG processor. The information is placed on the special file by the technology program. The file is later integrated by the DLG for possible placement of the information into the EDIN data base.

A series of four routines for printing the common types of data in a format readable by DLG are available. They may be called at any point in the calculation sequence for generating EDIN output. The format simulates the control directives format used in the DLG processor.

ADDREL - For printing real variables and arrays.

ADDINT - For printing integer variables and arrays.

ADDHOL - For printing Hollerith variables and arrays.

ADDLOG - For printing Logical variables and arrays.

The output is similar to the format of NAMELIST for one variable name only with any number of associated values. Each subroutine has the same calling sequence characterized as follows:

CALL ADDREL (LU, NAME, NUM, VALUE)

LU - Logical unit or special output file.

NAME - Desired name chosen by the analyst/programmer. It may be a stored name set by a Fortran data statement or can be set in the calling sequence as nHname.

NUM - Number of values in the array. For a single variable NUM=1.

VALUE - Internal variable or array name (starting location).

The subroutines for the other variable types have the same calling sequence. The primary difference among them is the format used for writing the variables and the special output file. Each output is a DLG control directive format. The name associated with the directive is set by a data statement in the individual subroutines. The data statement may be set at the time the technology program is modified. Usually it is desirable to use a name which is reminiscent of the application program name. The selected name may be precisely the same as the acronym used to execute the application program in EDIN. The reason for such a choice is that the directive name is stored in the EDIN data base. A print of the data base prints the last directive which updated each variable in the data base.

For most technology programs, the use of the software described above is adequate. However, certain programs generate data base information in a Fortran "DO LOOP." In these instances, the package (by itself) can not satisfy the EDIN requirement of separate names for different data elements and arrays.

The most convenient way to make this program and others of this type compatible with EDIN is to provide some name-generating capability with the applications program. Function subroutines which provide this capability can be called as illustrated below:

NAMGEN (NAME, K, J)

NAME = The desired root name.

I = Concatenated number occupying the first one or two BCD character positions beyond the root name.

J = Concatenated number occupying the second one or two BCD character positions beyond the root name.

An example would be:

```
NAM=NAMGEN (4HNAME,1,2)
```

In the above illustration, the name NAME would be extended by the BCD characters 1 and 2 concatenated to it and stored in NAM.

NAM=6HNAME12

A maximum of 6 characters may be generated. This limit is imposed by the word size limit for EDIN data base names.

Usually the NAMGEN function is used in conjunction with the NAMELIST simulator described above in the following manner:

```
CALL ADDREL(LU,NAMGEN(NAME,I,J),NUM,VALUE)
```

In the illustration, the name is generated within the calling sequence of the subroutine which prints the simulated namelist for the generated name.

PROGRAM UTILIZATION

DLG Usage

Control Statement.-

```
@DLG.DLG,options lfn.eltl,lfn.elt2
```

Option Specifications. -

I Source input will follow the processor card. Source output will be placed in eltl.

L Source input data will be listed.

O Source output data will be listed.

D Card cracking information will be listed.

E Solicitation and result of directives will be printed.

S List interrupt mode will be invoked.

M New data base files will be generated with this execution.

B Build option will be invoked. This option specifies that all data directives of the form:

```
'name name=value---
```

or

$name name=value---

This will permit the addition of data to the data base regardless of the directive name. Otherwise, only those data base variable names, which were previously defined in the data base, will be updated unless the data directive name is ADD or DEFINE.

The B option may not be invoked via the "ON" command. If desired, it must be present on the processor call card.

Syntax Definition. -

name	Must be six (6) or less alphanumeric characters and begin with an alphabetical character.
'(quote or prime)	The DLG delimiter. Strings that occur between pairs of delimiters will be processed by DLG. Strings external to primes will be passed "as is" into the output element.
-	The underline on a command indicates an optional character string which may be used as a directive.
value	Indicates a data base value in real, integer or hollerith format.
i,j,k	Indicates integer constants used in the directives.
elt	Exec 8 file element name in program file format.
lfn	Exec 8 logical file name in system data format.
text	Textual information.
[ ]	Indicates optional items on the line.

Descriptions of Control Directives. -

'ADD name' - Specifies that information will be added to data base.

'ADD name=value'

'ADD name=name'

'ADD name=value,value,---'

'ADD name=name,name,---'

'ADD name=name op name, name op value,---'

	+	Add
	-	Subtract
where op =	/	Divide
	*	Multiply
	**	Exponentiation

'CHANGE number=value' - Using the integer number 'number' as an index into the master common block, IDILOG, the current value is replaced by 'value.'

'COMMENT _____' - This is a null card and is discarded by DLG.

'CREATE name,DIRLEN=number,LENDES=number,LTOTAL=number' - The data of name 'name' is brought into existence on the data base file. Optional parameters are DIRLEN - the directory length (This should be a prime number.).

LENDES - Length in computer words of the description.

LTOTAL - Total size, in computer words, reserved for the data base.

'CHANGE' -

Example 'CHANGE 27=3'

Location 27 of the common block IDILOG will have its value replaced by an integer 3.

'COMMENT' - A null card. The delimited field is removed from the card. If the resulting card is BLANK, the card will be removed from the run stream.

'CSF @ Control Statement' - Specifies that an execution control statement will be processed using the standard CSF$ package. The following control statements may be used:

@ADD	@CKPT	@RSPAR
@ASG	@FREE	@RSTRT
@BRKPT	@LOG	@START
@CAT	@MODE	@SYM
@CKPAR	@QUAL	@USE

Example:

'CSF @USE 25, DBASE'

'CSF @ADD DUSEFIL.DLOG'

'CSF @QUAL B'

'DEFINE name=value,text' - Stores a textual description with the name in the data base directory. If the name is a new directory entry, the value is the number of data base entries allotted. Existing data is unaffected and new data is not added.

'DEFINE A, LETTER 1' - Stores the description, LETTER 1, with the name A.

'DEFINE B=10, BARRAY' - Stores the description, BARRAY, with the variable name B and allots 10 data base entries for B.

'FORMAT name=value/value, (Fortran compatible format statement)' Extracts freely stored data from the data base and places into the output elements in accordance with the given format.

'FORMAT A=6/3,(1X,3F15.3)'

The six items of A are output into the named element, 3 on a line through the (1X,3F15.3) format.

'INSERT name=value/value' - Specifies that binary coded information the SDF file name will be placed in the source output element in 14A6 format.

'INSERT A' - Entire file of data in A will be transferred to source output element.

'INSERT B=5-13' - Insert data from B from records 5 through 23.

'INSERT C=5*EOF' - Insert records from file C records 5 to the end-of-file.

Other Examples - 'INSERT A,B=5-23, C=5*EOF'

'name' - Specifies a simple replacement of named information with data base parameters or arrays.

'REAL'	Real parameter or array.
'INTEG'	Integer parameter or array.
'HOLITH'	Hollerith parameter or array.
'LOGICL'	Logical parameter or array.
'ARRAY(j)'	Real or integer element of an array, j must be a constant greater than 1. A value of j=1 will cause the transfer of all of j.

'ON name,name---' - Mode activation directive.

'OFF name,name---' - Mode suppression directive.

P or PAGDMP	Print card cracking information.
O or OUTDMP	List logical file 1 data.
N or INDUMP	List source output element.
C or CONTINUE	Activate continuation card option.
L or LIST	List source input information.
S or SPLIT	Interrupt mode.
E or EDIT	Edit mode (demand response to printer).

'PRINT name' - Specifies that data information will be printed.

'PRINT name=A,Z'	Print all information in name.
'PRINT name=n,m'	Print entries n through m alphabetically.
'PRINT name'	Directory and first data base entry of named data base.
'PRINT'	Directory and first data base entry of current 'USE' assigned data bases.

'USE' -

'USE A,B,C' - The data bases named will be circularly searched in the order given for variables used in replacements. All will be searched once before a NO FIND is declared. It should be noted that this command may cause very excessive SUP changes if not carefully used.

'UPDATE name' - Specifies that the named data base will be updated with the information which follows:

'UPDATE A' - Specifies that the data base A will be updated with the data which follows.

Examples

Example 1. - Data base creation.

```
@fn.DLG,GI   .RPT

 'CREATE AERO,LTOTAL=4000,DIRLEN=501'
                [INITIALIZATION
                    DATA       ]

 'CREATE STR,LTOTAL=6000,DIRLEN=557'
                [INITIALIZATION
                    DATA       ]

 'CREATE MASSP,LTOTAL=9167,DIRLEN=487,LENDES=7'
                [INITIALIZATION
                    DATA       ]

 'PRINT'
```

Example 2. - Data for DLG is in input stream.

```
@DLG,I  PANDATA
```

```
$IPANEL  IGEOM=5,$   <-------- Element PANDATA Constructed
  'UPDATE DBASE'               from this Data.
$INLIPS

  THETOX='DBARA'          • DBARA from Data Base.

$END
```

```
@ASG,T 8.

@ASG,T 11.

@XQT PANEL.OPANEL

@ADD PANDATA
```

```
@USE 14,NMLIST
@fn.DLG,I MAIN
```

```
NAMELIST/OUT/C
'UPDATE DBASE'
CALL GETA(A)
B = 'SMAXIS'
C = A**2+B**2
WRITE(14,OUT)
END
```

◄ Element MAIN constructed from this data.

- GETA is an element of BRARY.
- SMAXIS from data base
- Data on 14 goes to data base on next execution of @fn.DLG.

```
@FOR MAIN
@MAP,I DUM,MYPGM
 IN MAIN
 LIB BRARY
 END
@XQT MYPGM
```

- BRARY contains GETA

Example 3. - Data for DLG stored in element.

```
@fn.DLG NAME1,NAME2
@XQT PGM
@ADD NAME2
```

```
NAME1 Element
'UPDATE DBASE'
'DATA'
```

◄ Element NAME2 constructed from this data.

```
@fn.DLG NAME3,NAME4
@ADD NAME4
```

```
NAME3 Element
@XQT PGM
'UPDATE DBASE'
'DATA'
```

◄ Element NAME4 constructed from this data.

Example 4. - Linked Simulation.

```
@ASG,A DATA7                    • Assign permanent data base.

@COPY DATA7,DBASE

@fn.DLG EDIN.SIM,MODIN

@ADD MODIN
```

```
EDIN.SIM Element

@XQT PG1

   'UPDATE DBASE'
   'DATA'
@fn.DLG LUI.RPT,SUMRY

@ADD SUMRY
```

← Element MODIN constructed from this data.

```
LUI.PRT Element

 SUMMARY REPORT

 'UPDATE DBASE'
 'DATA'
@fn.DLG PIX,MODIN
@T.OIMAGE
@ADD MODIN
```

◀ Element SUMRY constructed from this element.

Example 5. - Generation of a data base status report.

```
BATCH MODE:
@USE DBASE,DATA7                • Assign permanent data base.
@fn.DLG,M EDIN.STATUS,STATUS
```

```
EDIN.STATUS Element

 STATUS REPORT

  'UPDATE DBASE'
  'DATA'
```

DEMAND MODE:

```
@fn.DLG,EI HISTORY
>'UPDATE DBASE'
> 'NAME'              • USER INPUT
   value1             • DLG RESPONSE
> 'ADD A=VALUE        • USER INPUT
>      B=V2'          • USER INPUT
> 'A  '    'B   '    • USER INPUT
   value2  value3     • DLG RESPONSE
> @EOF                • USER INPUT
```

HISTORY Element

value1

value2 value3

NOTE: The character (>) is used to denote a demand read request.

Example 6. - Conditional looping by combining the capabilities of EX8 and the DLG processor.

```
@fn.DLG LOOPELT,MODIN
@ADD MODIN.
@fn.DLG LOOPELT,MODIN
@ADD MODIN
@fn.DLG LOOPELT,MODIN
@ADD MODIN
@STOP:
```

LOOPELT Element

```
'UPDATE DBASE'
@fn.DLG A,AA
@XQT PGMA
@ADD AA
@fn.DLG B,BB
@XQT PGMB
@ADD BB
@SETC 'CONVG'
@TEST TE/1
@JUMP STOP
```

← Element MODIN constructed from this data.

• SET CONVG

CONCLUDING REMARKS

The DLG Processor has been developed particularily for the 1110 Exec 8 operating system and other computers with similar file manipulation capabilities. The processor operates on data structures at the same level (element level) as the Exec 8 utility processors and language processors. Further, the DLG was designed to take full advantage of existing capabilities. Beyond that, the DLG Processor extends current capabilities by permitting parameterization of any specified data element with information requests which can be satisfied by the DLG Processor at execution time from information stored in a stratified data base also accessible to the processor.

Data Base. - The data base is structured to permit the storage and retrieval of information of a parametric or matrix nature in real, integer, hollerith or logical type. The data base structures can be free data base structures, random access and tree structures.

Design Simulations. - The generation of a computer aided design study or simulation looks pretty much like a normal EX8 run stream. The execution of the DLG processor is interjected as required whenever a data base maintenance or retrieval function is indicated. Design loops are generated by the construction of partial run streams (@ADD files), which can be repetitively executed in an automated manner until the convergence criteria, (usually a data base variable) is met. All programs have access to the data base through the DLG processor. Therefore, any program (and/or the user) may control the convergence criteria.

Comparison with other Processors. - The use of the DLG processor is much like any of the language processors or utility processors on the EX8 system. Input data can immediately follow the processor card or can come from the source input element. The processed input is written into the source output element. The DLG processor contains a control directive language somewhat analogous to the @ED processor language but the control directives are designed for the purpose of maintaining data structures of various types rather than strictly card images. The directive language is patterned after the language structure in the DIALOG Executive System but with some significant differences and a considerable expansion in capability.

For example, entire data structures can now be searched, altered and generally manipulated by user commands. Data bases can be completely restructured at execution time to meet access requirements of a particular situation.

Compatible Developments. - The DLG processor is upward compatible with the GTM processor. GTM structured files can be read, displayed and reformatted by the DLG processor for other uses. Arbitrarily generated sequential binary file structures can also be interpreted and manipulated by DLG if the user knows the file format.

Reports of any nature, which conform to the element data format, can be generated, printed and permanently or temporarily stored on the mass storage media of the 1110. Another processor called CIPHER (reference 1) greatly expands the capabilities which now exist for report generation. The design of CIPHER permits compatible data handling with the DLG and GTM processors and indeed will operate upon the same data bases.

The existing report capability has been expanded to permit online edit and data base manipulation. The edit capability provides both demand response from DLG while simultaneously generating a permanently stored report.

REFERENCES

1. Glatt, C. R., Hirsch, G. N., Alford, G. E. Colquitt, W. N. and Reiners, S. J.: The Engineering Design Integrattion (EDIN) System. Aerophysics Research Corporation. JTN-11. 1974.