NASA CONTRACTOR REPORT

NASA CR-1976

NASA CR-1976

A SYSTEM OVERVIEW OF THE AEROSPACE SAFETY RESEARCH AND DATA INSTITUTE DATA MANAGEMENT PROGRAMS

Prepared by
NEOTERICS, INC.
Cleveland, Ohio
for Lewis Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • MARCH 1972

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
CR-1976		

4. Title and Subtitle	5. Report Date
A SYSTEM OVERVIEW OF THE AEROSPACE SAFETY RESEARCH AND DATA INSTITUTE DATA MANAGEMENT PROGRAMS	March 1972
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No. NEO 2040-711
	10. Work Unit No.
9. Performing Organization Name and Address Neoterics, Incorporated 2800 Euclid Avenue Cleveland, Ohio	11. Contract or Grant No. NAS 3-13341, NAS 3-14979
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

Project Manager, Charles M. Goldstein, Aerospace Safety Research and Data Institute, NASA Lewis Research Center, Cleveland, Ohio

16. Abstract

NASIS, the NASA Aerospace Safety Information System, is an interactive, generalized data base management system. The on-line retrieval aspects of NASIS provide for operating from a variety of terminals (or in batch mode). NASIS retrieval enables the user to expand and display (review) the terms of index (cross reference) files, select desired index terms, combine sets of documents corresponding to selected terms and display the resulting records. NASIS will allow the user to print (record) this information on a high-speed printer if desired. NASIS also provides the ability to store (save) the strategy (NASIS commands) of any given session the user has executed. NASIS has a searching and publication ability through generalized linear search and report generating modules which may be performed interactively or in a batch mode. The user may specify formats for the terminal from which he is operating. The system features an interactive user's guide which explains the various commands available and how to use them as well as explanations for all system messages. This "explain" capability may be extended, without program changes, to include descriptions of the various files in use. Coupled with the ability of NASIS to run in an MTT (multi-terminal task) mode is its automatic accumulation of statistics on each user of the system as well as each file. NASIS's ability to handle the storing of data was implemented to complement its retrieval abilities. A wide variety of files are handled with variable length fields and records which can be defined, created and maintained with no program modifications. File definition can be interactively achieved as well as in a batch mode. File loading can

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Management information system; Data storage and retrieval; Information storage and retrieval; Computer systems; Data base; File; Report generation; Linear search; Real-time; Multi-terminal task; MT/T; On-line; Range search	Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	63	$3.00

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

also be done on an interactive basis. This flexibility is possible because NASIS is based on a universal-record-format and a descriptor-driven file access. Whereas the maintenance is typically handled in batch mode, a correct facility is provided to allow the interactive creation of maintenance transactions. The original function of NASIS was the handling of scientific and technical information; many other uses for NASIS have been identified. For example, it has been used for project control, data set control and programming systems control. The NASIS system was designed for a virtual-memory, time-shared, paged environment. It is written primarily in PL/I with input/output modules written in assembler. The current implementation is running on an IBM 360/67 under the TSS operating system. The system was developed for the National Aeronautics and Space Administration's Aerospace Safety Research and Data Institute (ASRDI), Lewis Research Center by Neoterics, Inc.

INTRODUCTION

The data base management system herein described was developed to satisfy the need for a safety data base for ASRDI. Prior to the development of the NASA Aerospace Safety Information System (NASIS), information needs were satisfied by the RECON system which contained, among other items, safety items.

NASIS (the NASA Aerospace Information System) has been designed to service a variety of data management tasks within a large-scale, time-shared, virtual-memory, paged environment.

All design and implementation was done with the purpose in mind of providing a generalized system capable of handling many different files. Parameterized file creation and data retrieval were the cornerstones of the development work. It was a primary objective that the retrieval system would be operational on an IBM 360/67 under the Time Sharing System(TSS) and that the execution of the modules would be available in an interactive mode. Since the system was designed generally, provision was made to allow for evolutionary enhancements. The following overview defines the components of the NASIS system as it currently exists. It is likely that, while the basics of the system will probably withstand much of the evolution, the other functions may change with future enhancements.

In conjunction with this report will be later publications of two further reports of a limited distribution. The first, entitled "The NASIS System User-Oriented Reference Manual" (NASA CR - 72881), provides the NASIS user with the philosophy of operation and the details of utilization of the various NASIS commands and subsystems. The second report, entitled "The NASIS System Development Work Book" (NASA CR - 72882), not only encompasses much of this report and the Reference Manual but also documents the design specifications and data specifications of the NASIS system modules. Furthermore, the Work Book provides system testing, system maintenance, data base administration, and system documentation procedures.

The NASIS system performs a sophisticated information management function which encompasses the realm of information storage and retrieval. The NASIS system's structure is applicable to many other scientific and commercial information systems; consider the following applications:

- Personnel Skills Inventory

  Since data retrieval is accomplished via cross reference files (inverted indices) into common fields throughout the data base records (typically, a performance evaluation field), a personnel data base could be managed for job allocation, personnel advancement, etc.

- Medical Data Management

  Staff, patients, and facility utilization and allocation may more easily and efficiently be performed by hospital management through the assistance of the interactive information storage and retrieval structure of the NASIS system.

- Bibliographic Retrieval

  This type of information retrieval is a direct application of the NASIS system presented herein.

- Urban Information Systems

  Since urban problems have become directly related among city governmental departments, the data about a city's inhabitants (children, voters, welfare recipients) and its events (crimes, fires, ambulance emergencies) is of great value to the decision-making authorities. An interactive data retrieval function of this type could be satisfied by the application of the NASIS system.

Other potential applications include the following:

* Inventory Control
* Payroll Accounting
* Budgetary Control and Analysis

TABLE OF CONTENTS

TOPIC B.1 NASIS DESIGN OVERVIEW

TOPIC B.2 DATA BASE EXECUTIVE OVERVIEW

TOPIC B.1 - NASIS DESIGN OVERVIEW

I. SYSTEM OVERVIEW

A. Design Objectives

To facilitate the definition of this project, certain design objectives were established. Not only did these objectives serve as a means for defining the scope of the project, but they also provided a basis for the overall design of the system itself. The design objectives were:

1. The design of the data base and the system should be modular so that it facilitates change and expansion.

2. The data base should be capable of being accessed by a number of concurrent TSS users.

3. The data base should include the framework necessary for the creation and use of inverted indices.

4. Access to the data base should be controlled so that the integrity of the data base can be maintained.

5. Batch maintenance facilities should be provided for the data base in such a manner that maintenance can be done concurrently with retrieval.

6. The maintenance facilities should provide for automatic verification and translation of the input transactions, as well as a complete audit trail of the maintenance results.

7. Interactive retrieval capabilities should be provided which duplicate those currently available under NASA's RECON (REmote CONsole) system and also the desired extensions mentioned in the RFP (Request for Proposal, Aerospace Safety Research and Data Institute) system.

8. A standard facility should be provided for the definition of new files to the system and for the inclusion of the new data files in the system data base.

9. The owner of a given set of files should have the ability to control read access to them, on a field and/or file basis; he should have read/write access to them himself.

10. It should be possible to define both variable-length records and variable-length fields whenever wide variations in data quantity are anticipated.

11. Concurrent file maintenance and retrieval should be premitted. Multiprogrammed, concurrent maintenance on the same file should be precluded in this design.

12. The data base should be protected by file backup and recovery procedures that should be able to insure the cessation of maintenance at a known point and the prevention of any further maintenance during the time that the file is being dumped.

B. Design Restrictions

In designing a system, especially one as complex as this, there were special restrictions of which the designers were aware. These restrictions differed from design objectives in that they were characteristics of the system or its environment which limited the alternatives to many of the design questions. The restrictions, in this case, were divided between operating system (TSS) restrictions and application restrictions.

The TSS restrictions were:

1. The system should run as a "user" under TSS.

2. The design should be built around the system standard and the concepts of TSS.

3. The TSS data-sharing facilities should be used as the basis for providing data file security.

The application restrictions were:

1. The programs involved should be written in PL/I, whenever possible.

2. The files should require each record to be uniquely identified and accessed by the identifier, so that the virtual indexed sequential access method (VISAM) is the most logical file organization technique to use.

3. Constructing or expanding the dataplexes must be possible by converting and including files already in existence elsewhere.

C. Executive Overview

The primary component of any information storage and retrieval system must be the data base. For the NASIS system, an integrated set of VISAM files (data sets) constitutes the data base. Reference data from a particular source is stored in a number of associated linear files. The information in these files is indexed by the contents of various control fields and the results maintained in a set of cross reference files (inverted indices). Each data set in the data base has its fields described for format and content in a special descriptor data set. The data in the descriptor data set is used by the system executive routine to access and process the contents of a file.

The executive routine for the NASIS system controls the access to the data base by the processing programs. It provides to the processing program access to the information in a data set on a field basis, by field name, or on a record basis, by key or sequential. It uses the descriptors to provide automatic verification and conversion of the data as it is processed by the various programs. The executive provides a PL/I source level interface for the processing programs.

D. Mainline Overview

1. Design Objectives

The mainline routines control internal and external data manipulation. The objective was to provide a general, modular control function which is fast and may be easily altered or expanded, as appropriate.

2. Considerations

Each mainline must run as a user under TSS, but must be available for sharing by numerous IDs. Each mainline routine must be accessible only by those persons authorized to use the routines. This restriction is further qualified by whether or not access to the particular data is allowed in the mode which the user requests.

3. Overall Structure

NASIS is a data management system with interactive capabilities. To most people, an interactive system implies an on-line query type system of limited capabilities. NASIS, on the other hand, represents a powerful data management system enhanced by, but not limited to, interactive capabilities. The inclusion of interactive capabilities as a design requirement has, of course, affected the entire system design.

NASIS is monitored and supervised by a data base executive and a terminal handler. The former module controls all access to the data base via various file and field security checks, performs all reads and writes to the files, automatically updates cross references (inverted indices) during anchor file maintenance, and uses the descriptors to validate and convert data on input to and output from the files. The terminal support modules provide data transmission facilities between the NASIS system and TSS and specialized support of graphic terminals which are not supported under TSS.

At present, eight mainline actions required to drive the system have been implemented. Most op-

erations require access to the descriptor files by the executive and perform the following actions:

a. Descriptor Editor

The Descriptor Editor provides a means of creating descriptor files. Descriptor files consist of the field descriptors which describe the records of a set of files (dataplex). No attempt is made to load any data into the file at this point.

b. Load/Create

The loading of a file is the accumulation and storage of data upon some physical device. Loading can be accomplished by processing ADD transactions through the Maintenance Mainline. Also, a special Load program is available to accept an externally-defined file as input, allow a user-written routine to break down the fields, and use the data base descriptors to enter the fields into a NASIS data set.

c. Maintain

Maintenance consists normally of adding, deleting or changing data which is incorrect on a given file or set of files (dataplexes). In the NASIS system, this has been provided for on the field level through the use of transactions. Records can be wholly deleted or created by key. Fields can be wholly added, changed or deleted or partially changed based on specified contextual character strings.

Once it is verified that no other maintenance is active on the same file, the file is accessed in the update mode. Any inhibited maintenance runs are terminated and not queued.

d. Backup

Backup is a utility program which copies the prime file or files to other secondary devices to preclude the inadvertant destruction of entire files. Backup requires that maintenance be inhibited and that no maintenance be in progress. The data records may then be read sequentially and copied to the backup medium. The descriptors are not required for record processing because the records are known to the system.

e. Restore

Restore is a utility program which supplements the Backup utility by providing a means of copying data from some secondary device to a primary device to restore to usefulness any data sets (files) which have been destroyed.

Opening the receiving file in the output mode locks out all access until the file is closed, thereby precluding premature access to the file. The operation consists of copying the backup file into the data base, unchanged from its original format.

f. Reorganize

Reorganization is a utility which changes the internal structure of a file whenever it is necessary. Reorganization requires that maintenance be inhibited for the entire operation. It consists of running Backup followed by Restore.

g. Purge

Purging a file requires that Backup be run if future access to the data is required. Both the descriptors and the file can be erased to effect the Purge function.

h. Retrieve

The retrieval aspects of NASIS are completely on-line and interactive. The central facility in retrieval is a parameterized command system which permits data selection based on either auto-

matically-maintained cross reference files (inverted indices) of certain fields or a linear search of all or part of any file. Output is available to either an on-line terminal (including CRT devices) or an off-line printer using a generalized report and format generator which is also part of the command system.

The primary emphasis on the system, data retrieval, is handled by a specialized command subsystem. The command language can be categorized into three major classifications: search, output manipulation, and transaction generation.

The commands EXPAND, SELECT, and LIMIT are available for use in searching the cross reference files (inverted indices) for particular information (i.e., the "browsing" capability). SEARCH and EXSEARCH are used to retrieve data from any field in the files whether indexed or not. STRATEGY allows the user to review the current retrieval strategy he is executing (in effect, a synopsis) or to observe strategies which were previously defined and have been permanently stored. To preclude having to redefine standard strategies, RERUN is available to allow re-execution of stored strategies. Recovery capabilities in the event of computer or system malfunction are enabled by the RESTART command.

Screen manipulation and output formatting, both screen and hard copy, are the functions of the second category of commands. REVIEW (DISPLAY) causes data set displaying on a remote terminal in either one of four fixed formats or a user-defined format. RECORD (PRINT) generates the appropriate output on the high-speed printer. User-defined formatting is available through the FORMAT command for both the printer and the screen. PAGE is used to manipulate the screen output and KEEP and SAVE are specialized commands providing temporary storage capabilities for selected data or CRT screen images.

The CORRECT command constitutes the third category of commands. Although it is technically part of the retrieval system, CORRECT is more logically associated with maintenance since it provides the means for defining transactions used in updating the files. Interactively, the user can enter changes to any field in an existing record on a contextual basis, add or delete an element on an entire field, or delete the entire record.

Retrieval requires read access only to both the descriptors and the file.

A graphic representation of these functions is shown in Figure 1, "System Overview". Each of the capabilities defined above is described seperately under the appropiate sections in the remainder of this report.

4. Data Control

The mainline logic routines require occasional read access to several key indicators. The setting and testing of these indicators is performed by the executive in the case of the MAINTAIN and STATUS indicators and by the mainline in the case of the RETRIEVE and ENABLE indicators. It is the responsibility of the executive to insure that the value of an indicator accurately reflects the current situation. Otherwise, an inaccurate reading can be introduced by the asynchronous operation of two programs using the same physical file. These four indicators are:

a. Data Present (RETRIEVE)

This variable is used to indicate a null file. This condition exists when a descriptor has been defined, but the data file has not been loaded.

b. Inhibit Write Operations (MAINTAIN)

This variable is used to indicate file loading. All other functions except retrieval are prohibited.

c. Maintenance-in-Progress (STATUS)

This value indicates whether maintenance is currently in progress, thus precluding any other operation except retrieval.

d. Descriptor Ready (ENABLE)

This variable is set on when the descriptor file is ready for use.

II. DATA BASE

A. Overview

1. Files

The NASIS system revolves around series of file sets which are called dataplexes. A dataplex is a set of files normally consisting of an anchor file, and one or more inverted files (cross reference files). Each file is composed of records and each record is composed of fields. There exists another file in this file set which is the descriptor file. It describes the records of each of the files in a dataplex. The descriptor is at the field level and therefore all access to any or all files of a dataplex can be at the field level as well as the record level.

The entire data base for the NASIS system consists of several components:

a. Field Descriptor Files

b. Linear (Bibliographic) Files
   i. Anchor
   ii. Associated

c. Inverted Index Files (cross reference files)

d. Thesaurus File(s)

Most of the various data sets in the data base are organized as variable-length record VISAM data sets; the descriptor file is a region data set. Although several mainline programs in the system manipulate both input and output, a generalized executive program handles all I/O interfacing between the VISAM access method and the user mainlines.

To avoid having to define a pertinent file in each program using it, each file is described by a set of field descriptors. These descriptors then serve as the source of all information required by the executive to perform I/O operations and validation on the file. Since every file has a set of descriptors defining it, the descriptor files, themselves, are defined by a set of field descriptors. This latter set, the descriptor descriptors, is defined within the confines of the executive, thus providing the executive with a means of interpreting the descriptors and the various data sets.

In describing the data base and the interrelationships of the files, a set of reference names are specified to identify the various levels of data within the data base. An ELEMENT is the lowest defined unit of logical information. A FIELD, the next higher level, may or may not contain multiple ELEMENTs. FIELDs are associated to form RECORDs. Each uniquely-defined TSS data set is comprised of multiple, keyed RECORDs. To this point, terminology has been limited to those terms defining intra-file components.

An independent source of information, when converted to the system, is represented by three types of files: anchor, associated and cross reference (inverted index). The anchor and associated files are both linear files, either of which may contain any field in the records. However, the information in the two types of files is mutually exclusive except for the key. Whereas the records of the anchor file and the associated file are physically seperate, they are logically one record (i.e., the associated record is a logical extension to the anchor record). These two files are referenced as a FILEPLEX. Any number of inverted indices may be associated with one FILEPLEX.

The cross reference files (inverted indices) for one FILEPLEX, when taken together, will be called an XPLEX. The FILEPLEX and the XPLEX, together, will be referred to as a DATAPLEX. See Figure 2, "Sample Data Base".

The Thesaurus Files are part of the entire data base, but are not part of any DATAPLEX. Instead, although there are no associated or inverted index files affiliated with thesauri, the Thesaurus files, by default, are considered to be a separate DATAPLEX.

The remaining components of the data base, the Field Descriptors, are themselves files but are always referenced and used in conjunction with one of the other files. Thus, the term UNION has been devised to associate a descriptor data set with the file it describes. For example, the NSIC anchor UNION would imply the NSIC anchor file and its descriptor file. The term, UNION, however, will also be extended to other levels of the data base. For example, the NSIC DATAPLEX UNION includes all linear, inverted index, and descriptor files associated with the NSIC information source.

2. Record Format

To promote standardization and uniformity of the data base components, it was decided to define a record format which would be used for every record in the data base. See Figure 3, "Universal Record Format". Several factors had to be considered in preparing and evaluating the alternatives. The considerations included:

a. Although all records must be variable-length, certain fields would be fixed-length.

b. Any of the fields, either fixed- or variable-length, could be devoid of data; i.e., null fields.

c. A field could contain a certain number of elements but that number may not exceed a maximum allowable number.

Typically, in fixed-length records, a field can be located by accessing its fixed high-order position. However, since fields can be variable-length, fields must be accessed by either an identifier attached to each field or a combination of relative field position and associated lengths. To reduce space overhead, it was decided to split each record into a fixed portion and a variable portion. Thus, within the fixed portion, fields could be accessed by the high-order position while the relative position and length attributes could be used to find a variable-length field.

Thus, the first part of each record contains all fixed-length fields and is, itself, split into three logical breaks. The first section of the base portion, as it is called, is a four-byte TSS-maintained field indicating the length of the entire record (in binary). Following the record length field will be the key (unique identifier) of the record. The third section contains all the remaining fixed-length fields. The byte length of all fixed fields, including the four-byte record length, is called the base length of the record and is recorded in a header record of the appropriate descriptor file. Since null fixed-length fields can exist, any such field is carried on the record as a field of all binary zeroes except for a 1 in the highest-order bit.

The remainder of each record, following the base portion, consists of the variable-length fields. To provide an easy ability to locate the proper field, each field could be preceded by a one-byte unique identifier. However, such a designator would require much storage utilization because of the large size of the files. A better solution to the location problem is to define a relative field number, relative to the first variable-length field, within the field descriptor. Since each variable-length field must be preceded by a length indicator, any field can be accessed by using the field's relative number in conjunction with the appropriate field lengths.

The length indicator for each variable-length field is a two-byte designator immediately preceding the field. The maximum field length is, therefore, 65,536 bytes. However, since TSS does not support spanned records, the true maximum field length is 4096 bytes. While a null variable-length field contains no data, the length indicator still exists and contains a length of 2 (the length of the indicator, itself).

Some fields within the variable portion may be composed of multiple, identically formatted subfields, or elements. As an example, the author field of a record may contain three different authors. These elements may be either fixed-length or variable-length. Recognizing proper elements, for maintenance purposes, can be achieved by either identifying each element by a designator attached to the element; positioning the elements in static relative locations, providing a null designator for missing elements; or omitting any kind of indication and forcing maintenance to reference the proper element by value.

Attaching a unique identifier as part of each element would be space consuming because of the number of elemental fields (e.g., indices) and, thus, was eliminated. Because a maximum number of elements must be specified for each field and because a great number of elements may be missing, or null, providing null indicators also would become space consuming. Maintaining static relative positions of elements thus would not be required. Updating by value provides the most flexibility in data management and also minimizes user requirements for altering elemental items.

By definition, elements can not exist within the base portion of the records since they would have to be fixed-length and can be identified separately as fields, anyway. However, elements can be either fixed-length (e.g., keys in the inverted indices) or variable-length. All elements of the latter type are preceded by one-byte length indicators (maximum length of elements is 256 bytes). However, contrary to the method of handling null fields, the length indicator is not retained for null variable-length elements. With null fixed-length elements, no space or reference is reserved. In both cases, any deletion of intervening elements causes a left-justification within the field of the remaining elements. By definition, an elemental field having all null elements is defined as a null field and contains solely the two-byte field length indicator.

3. Naming Conventions

All fields are defined by alphanumeric names not exceeding eight characters in length. The names must not begin with a number. Any field name requiring fewer than eight characters must be left-justified with spaces on the right. Each field name must be unique within the dataplex to which the field belongs.

File names will be treated as seven-character strings of alphanumerics. However, the last character of each file name is internally assigned while the first six characters are user defined. The user's file name also may not begin with a numeric. It must be left-justified and any remaining characters, up to six total, must be filled with dollar signs ($). Effectively, the user name becomes the name of both the anchor file and the dataplex containing that file. In addition, the six-character name is the base name for all other linear files and the inverted index files. The Descriptor Editor is responsible for assigning a one-character suffix for all files.

The assignment of the final character of the internal file name is according to the following algorithm:

a. For an anchor file, the character is a space.

b. For all other linear files, the character is numeric and sequentially assigned starting at zero.

c. For all inverted index files, the character is alphabetic and sequentially assigned starting at A.

d. In both the last two cases, should a file be purged, the character assigned to the seventh position is not reassigned. Therefore, gaps can exist in the numbering sequence.

Descriptor file names are internally derived by qualification of the name of the file which the descriptors define. By prefixing the character '#' to the anchor file name, a descriptor file can be properly identified (e.g., #STAR$$1).

In any program, any field in an anchor file can be referenced by the six-character user file name and an eight-character field name. A field in any of the associated linear or index files can also be ob-

tained by stating the six-character file name and the proper field name. While there is no difference, externally, in the referencing of the two types of fields, the executive handles the calls slightly differently, based on information contained about the fields in the descriptors. A program reference to a particular descriptor file is accomplished by specifying the seven-character file name preceded by the special character, '#'.

The actual data set names are qualified names. This helps to insure the integrity of the files. The first segment of the qualified name is the file owner's identification. The second segment of the qualified name is the six-character anchor file name. The last segment is the seven-character field name with suffix. For example:

OWNER-ID.STAR$$.STAR$$1

4. Security

Each file in the system has an owner who may be either an individual or a group. The owner of any file will have the ability to create or modify the file or its descriptors. However, the owner can share the use of his file with any other individuals or groups. In this case, the sharers are permitted access to the file on a read-only basis.

In addition to the file-level protection, a field protect feature is built into the descriptors and is utilized by the executive. By attaching a list of security codes to each field in the descriptor file, an owner can prohibit field access on any field to anyone having sharing capabilities on that file. Of course, if a user does not have shared access to a file, he thus has no access to the fields in that file.

The security precautions, as mentioned, are under the control of the system executive. Any access to the data base through any of the system modules must pass through the executive and is subject to a check of the security classification.

B. Field Descriptor Files

A field descriptor is a set of values which defines the attributes of a data field within a data set. For each data set in the system, another data set, containing a set of field descriptors, exists to define all the fields associated with the first data set. Each descriptor is a record within the file of descriptors. Although the descriptor files are essentially VISAM-oriented, they are all, in reality, organized as one region data set in which the dataplex name serves as the region name. The key to the records is the name specified in Table 1, "Descriptor File Record Format".

In the table, the first column is merely a reference number to each field. The second column is the descriptor field name; column three indicates either the field's location, in bytes, or its relative location (variable fields). The next two columns represent the field mode and its recording type, respectively, and the last column is a descriptive comment.

While each descriptor file contains detailed field information, certain file indicators and reference data must be stored for each data set in the system. To provide a means for storing this file-oriented data, each descriptor file has a header record, defined by a key of binary zeroes. The header record is 65 characters long, corresponding to the base length of the descriptors, and is defined in Table 2, "Descriptor File Header Format". The column contents are the same as for the previous table.

Although the header information is created and modified via the Descriptor Editor, the various indicators are changed by the Executive on command from the mainline modules. Thus, the accessibility of any data set varies dynamically depending on the action being executed on the file.

While the descriptors for the associated files and the inverted index files reference only the fields contained in those files, the descriptors for the anchor file are more comprehensive. In addition to defining the fields actually contained in the anchor file, the descriptor file also contains dummy field descriptors which reference fields not present in the anchor file, but contained in the associated and index files. In a sense, the anchor descriptor file serves as a dataplex descriptor file since every field contained in the dataplex is referenced there, either by direct definition or by dummy descriptors. The key of a dummy descriptor in the anchor descriptor file is the same as the key of the actual descriptor, for the same field, in the associated file

descriptor data set.

Since descriptors exist to accurately define the various data sets in the system, a change in the structure of any file necessitates a change in at least one field descriptor for that file. All maintenance of existing descriptors and creation of new ones is handled by a separate routine called the Descriptor Editor.

Through the Descriptor Editor, the user may create or modify any descriptor files which he owns. The user, however, need only access the anchor descriptor for any change to any of the dataplex files. The Descriptor Editor is responsible for taking the appropriate action on the anchor descriptors and extending any necessary changes to the proper associated or inverted index descriptors (using the anchor dummy descriptors as pointers).

C. Linear Files

Each source file which is to be loaded into a NASIS dataplex consists of records with uniquely defined fields. The person responsible for the loading of the file first defines to the NASIS system how he would like the resultant dataplex to be formatted. The actual loading process reads the source file and writes the dataplex.

Primarily, the file loader first decides where he would like various fields of his file to reside. He has a choice of either on an anchor file or on an associated file. In reality, the records of the associated file represent physical extensions to the anchor file records. There are two reasons for the placement of data on associated files: one is to prevent the anchor file records from exceeding 4000 bytes in length; the other is to isolate on a separate file those fields which are rarely accessed.

The anchor file is considered the prime file through which all other files are referenced. The other files are associated files. The entire set of these linear files is named the fileplex.

It should be mentioned again, for the sake of clarity, that all to the files of a dataplex are VISAM files, and that all of the records of each of these files has a universal record format.

One of the benefits of having the ability to split a logical record into several physical records on different physical files is that these files may exist on different devices, thus eliminating any constraints any one device might impose upon data storage.

The file names of all associated and inverted index files for the same dataplex are cross-referenced in the anchor descriptor file. It follows, therefore, that no inverted index, or associated file may exist unless the corresponding anchor file exists.

The associated files, because of their potential size and the lessened need for extremely fast retrieval, could reside on other high-volumn slow-speed devices. No field can be defined as belonging to both an anchor file and one of its associated files.

A thesaurus file is a special case of a linear file. It is not an inverted index and is an independent file. For all practical purposes, a thesaurus file can be considered a one-file dataplex and is treated as one.

D. Inverted Index Files

Index files are created to facilitate searching entire anchor files for particular kinds of information. For each dataplex, any number of inverted index files may exist, each one corresponding to a field in the fileplex.

The index files are VISAM data sets. The key for each file is derived from the field upon which the index is based; i.e., the key of the author index is the author name, itself. A record in any of the files consists of the key field and at least one anchor record key where each anchor record key will identify one record in the anchor file. Since any one record in an index can reference any number of anchor records, the lengths of the index records are variable.

The creation and maintenance of the inverted indices is completely under the control of the Executive. No external maintenance transactions are allowed to affect the index files. To accomplish the internal main-

tenance, each field requiring an index has, in its descriptor, the name of the appropriate index file. The name is prepared and inserted in the descriptor at the time of the creation of the descriptors for this dataplex (via the Descriptor Editor).

All the inverted index files associated with one dataplex are called, collectively, the XPLEX. Since the index files are totally subordinate to the anchor file in terms of creation, no index files can exist unless the corresponding anchor file already is present. It follows, thus, that deleting an anchor file implies deleting the entire dataplex, which includes the XPLEX.

REF. NO.	NAME	OFFSET BYTE(S)	MODE	TYPE	COMMENTS
1	RECLEN	0-3	Fixed	Binary	Length of entire descriptor in bytes, including itself (TSS maintained)
2	KEY	4-18	Fixed	EBCDIC	Identifies for this descriptor. Contains file and field names.
3	FLENAME	4-10	Fixed	EBCDIC	Seven-character file name for these descriptors.
4	FLDNAME	11-18	Fixed	EBCDIC	Name of the field within file.
5	INVFILE	19-26	Fixed	EBCDIC	Name of inverted file which indexes this field; blank, if none (Name obtained internally)
6	ASSOCFIL	27-34	Fixed	EBCDIC	Name of associated linear file which contains this field; blank, if none (Name obtained internally).
7	GENERCRT	35-42	Fixed	EBCDIC	Name of routine to be used for testing type of input characters (numeric, alpha, etc.); blank if none.
8	VALIDRTN	43-50	Fixed	EBCDIC	Name of routine to be used for special validation or conversion of input data; blank if none. Uses argument in VALIDARG.
9	REFORMAT	51-58	Fixed	EBCDIC	Name of routine to be used for any necessary output reformation or conversion (pack, unpack, etc.); blank if none.
10	OTHER	59-63	Fixed	EBCDIC	Currently N/A-blank
11	FLDPOSIT	64-65	Fixed	Binary	**Fixed fields** (FLDMODE=0) -Displacement of high-order byte of field from the start of the record **Variable fields**(FLDMODE=1) -Field number relative to the start of the variable portion of the record.
12	FLDLNGTH	66-67	Fixed	Binary	**Fixed-Byte fields**(FLDMODE=0, FLDUNITS=0) -Field length in bytes. **Fixed-Bit fields**(FLDMODE=0, FLDUNITS=1) - Bit position (0-7) of this switch within SWITCHES; **Variable fields**(FLDMODE=1)- (N/A),0.
13	MAXNOELT	68-69	Fixed	Binary	Maximum number of elements contained within field (0, if field contains no elements or is fixed).

Table 1. Descriptor File Record Format

REF. NO.	NAME	OFFSET BYTE(S)	MODE	TYPE	COMMENTS
14	ELTLNGTH	70	Fixed	Binary	**Fixed element** (ELTMODE=0) -Element length, in bytes; **Variable element** (ELTMODE=1)-Maximum element length, in bytes.
15	SWITCHES	71-72	Fixed	Binary	Contains all bit switches in the descriptor. (All switches are redefined individually below).
16	FLDMODE	71	Fixed	Binary	(Bit 0)-Format of field; 0=Fixed, 1=Variable.
17	FLDUNITS	71	Fixed	Binary	(Bit 2)-Unit of length for field; 0=Byte, 1=Bit.
18	FLDALIGN	71	Fixed	Binary	(Bit 4) Justification of data within fields: **Fixed fields** (FLDMODE=0), 0=Left, 1=Right. **Variable fields** (FLDMODE=1) - N/A,0.
19	INVERTED	71	Fixed	Binary	(Bit 6)-Indicates presence of inverted index file for this field: 0=No File, 1=File.
20	ASSOCIAT	72	Fixed	Binary	(Bit 0)-Indicates whether this field is contained in an associated file: 0=No File, 1=File.
21	ELTMODE	72	Fixed	Binary	(Bit 2)-Format of element within field: 0=Fixed, 1=Variable.
22	OTHERSW1	72	Fixed	Binary	(Bit 4)- N/A, 0.
23	OTHERSW2	72	Fixed	Binary	(Bit 6)- N/A, 0.
24	VALIDARG	Fld 1	Var.	EBCDIC	Argument to be used with VALIDRTN (Test pattern, limit, etc.); blank if none.
25	SECURITY	Fld 2	Var.	EBCDIC	Multi-element field containing codes which identify the level of security attached to this field. Elements are eight bytes long with a maximum of sixty-four per field.

Table 1. Descriptor File Record Format

REF. NO.	NAME	OFFSET BYTE(S)	MODE	TYPE	COMMENTS
1	RECLEN	0-3	Fixed	Binary	Length of header record, in bytes, including itself(TSS maintained).
2	KEY	4-18	Fixed	EBCDIC	Identifier for this descriptor. Contains file and field names.
3	FLENAME	4-10	Fixed	EBCDIC	Seven-character file name for this descriptor.
4	FLDNAME	11-18	Fixed	Binary	Contains binary zeroes.
5	RETRIEVE	19	Fixed	Binary	(Bit 0) - Defines whether the file can be read; 0 - read allowed, 1 - read prohibited.
6	MAINTAIN	19	Fixed	Binary	(Bit 2) - Defines whether the file is being loaded. 0 - file not loading; all functions allowed. 1 - file loading; only retrieval permitted.
7	STATUS	19	Fixed	Binary	(Bit 4) - Indicates whether maintenance is in progress. 0 - maintenance not in progress; all functions allowed. 1 - maintenance in progress; only retrieval permitted.
6	ENABLED	19	Fixed	Binary	(Bit 6) - Defines whether the descriptors are complete; 0 - yes, 1 - no (can not use them).
9	RECORDCT	20-21	Fixed	Binary	Contains the number of descriptors for this file.
10	BSELNGTH	22-23	Fixed	Binary	Length of the fixed portion of a record, including the RECLEN field.
11	FILETYPE	24	Fixed	Binary	Classification of file 0 - Thesaurus 1 - Anchor 2 - Associated 3 - Inverted index
12	INDEXCNT	25	Fixed	Binary	Identifier for the last assigned inverted index file.
13	LINEARCT	26	Fixed	Binary	Identifier for the last assigned associated file.
14	REMAIN	27-69	Fixed	EBCDIC	N/A - blank.

Table 2. Descriptor File Header Format

FIGURE I. SYSTEM OVERVIEW

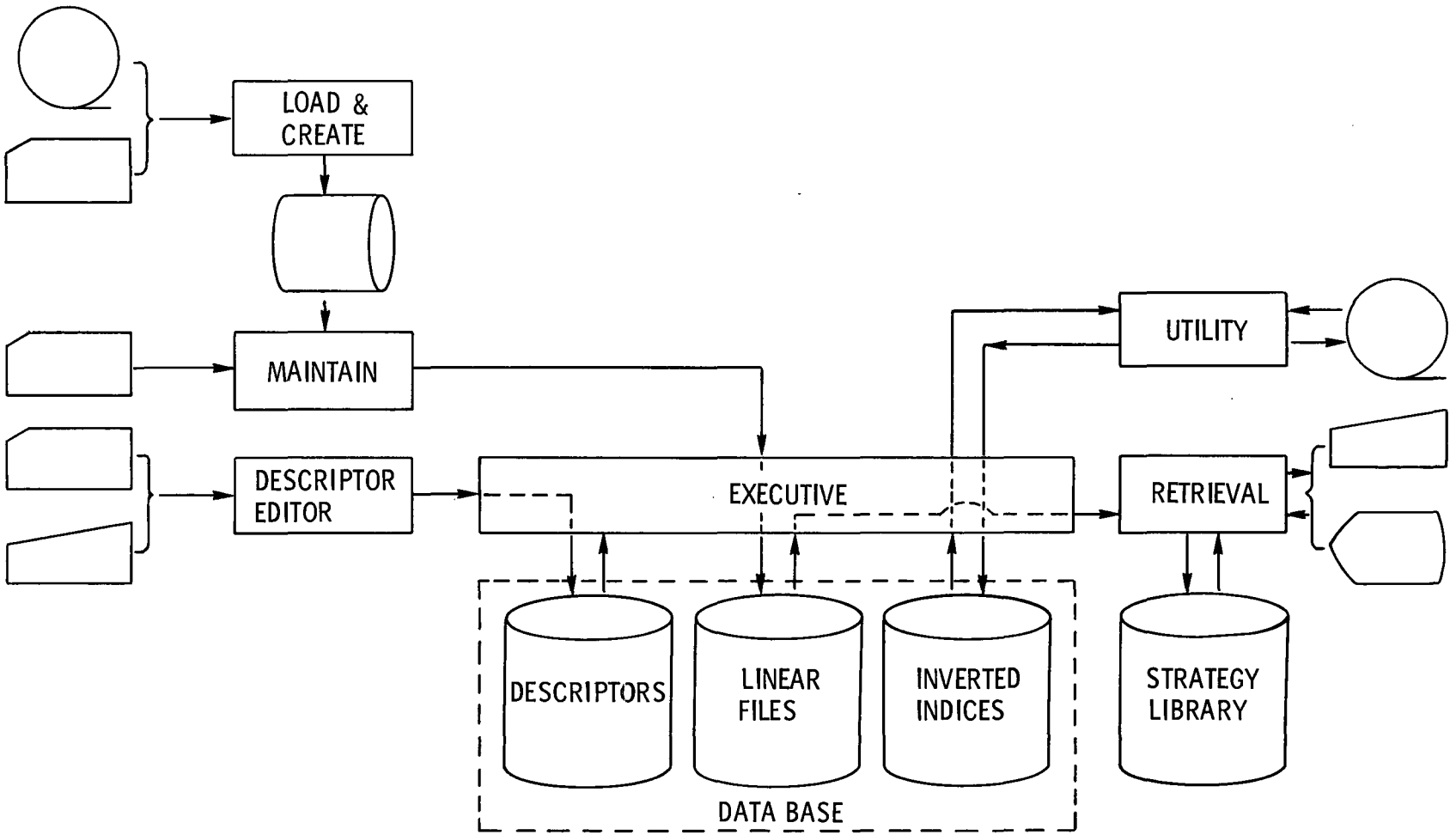


FIGURE 2. SAMPLE DATA BASE

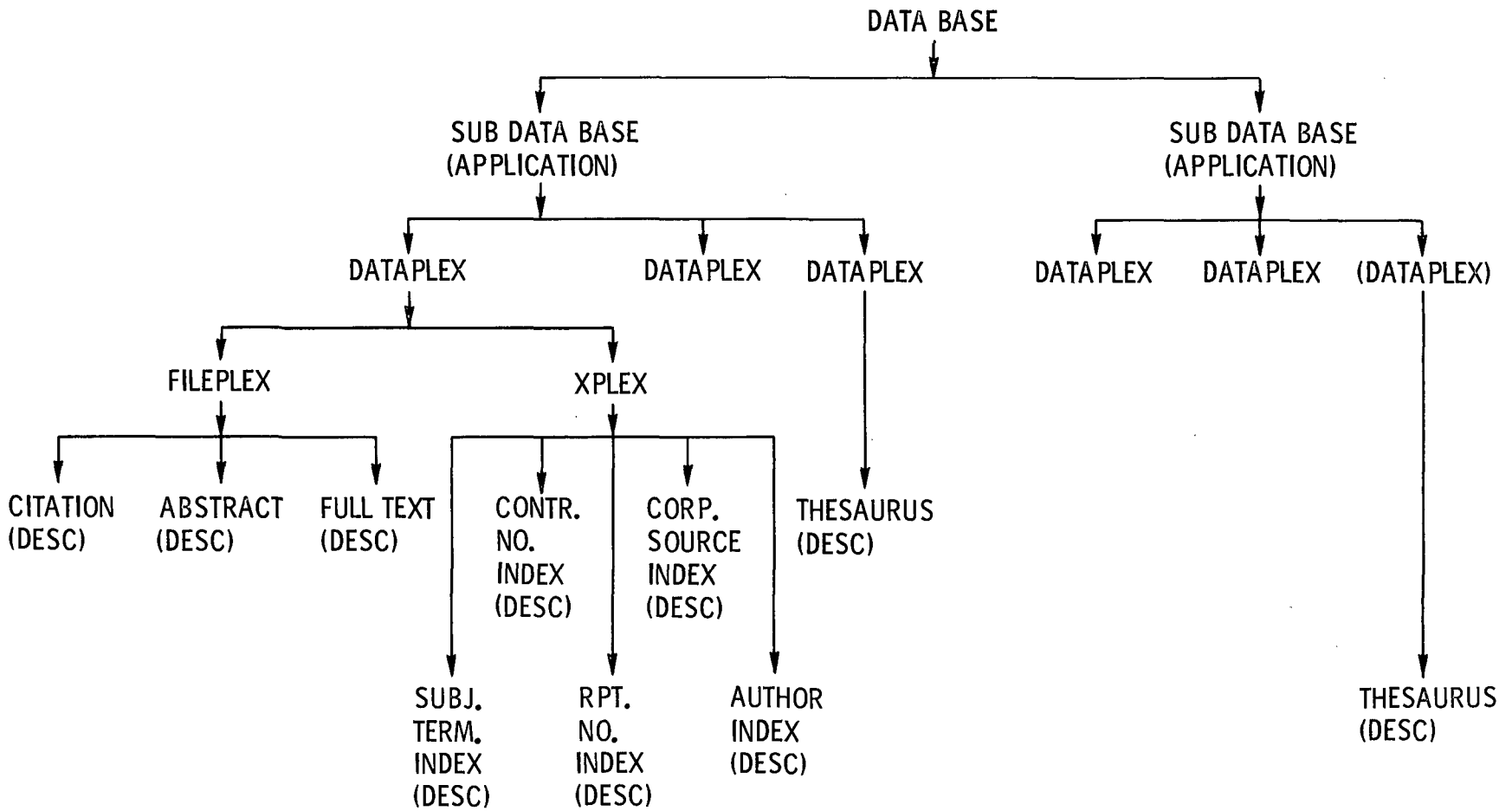
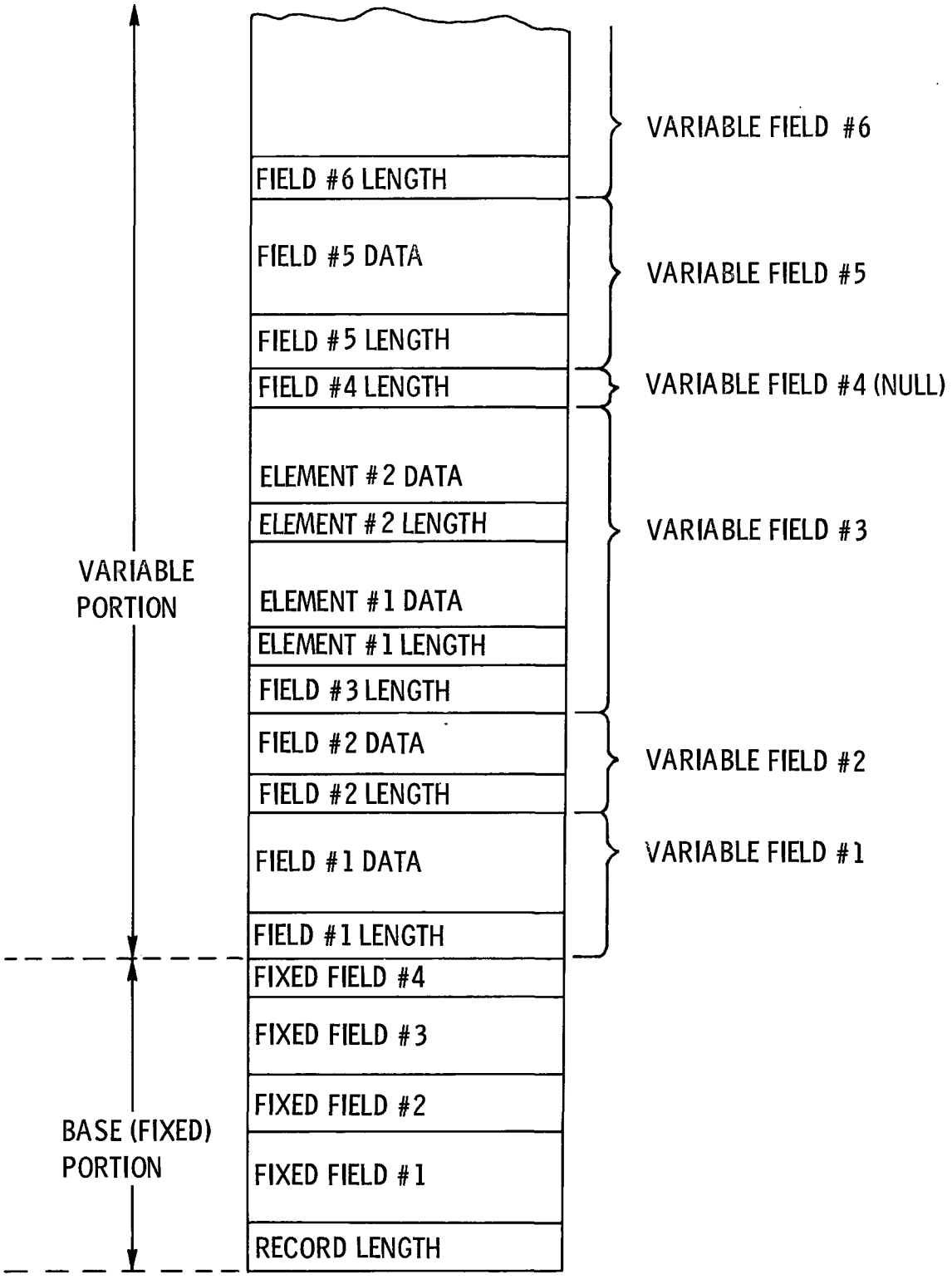


FIGURE 3. UNIVERSAL RECORD FORMAT



TOPIC B.2 - DATA BASE EXECUTIVE OVERVIEW

I. OBJECTIVES

A. Purpose

The data base executive subsystem (DBPAC) provides high-level (physical) access to the data base as a centralized service for mainline programs and, at the same time, guards the integrity of the data base for its owner. The following paragraphs describe the objectives more fully, and Section II of this topic describes the approach we propose to satisfy the objectives.

B. Data Base Security

DBPAC allows the owner of a data set complete access to his data set. It also protects the data set from other users; however, in certain situations, the owner of a data set will desire the ability to share his data set with other users. DBPAC allows data set sharing, on a read-only access basis, by owner assignment. This security is set on both the file and field level.

To further enhance the prevention of accidental data set destruction, DBPAC, through the use of four switches, recognizes and reacts to several situations. It precludes concurrent maintenance on an individual file. It does not allow retrieval on a partially-created file. It does allow maintenance concurrent with retrieval, but it will disallow maintenance during reorganization and vice-versa.

Further protection on the file level is available for cross-reference files (inverted indices). DBPAC has the ability to completely maintain the inverted index files for a given field. Then, by allowing read-only access of the inverted indices, inadvertent file destruction is precluded.

On the field level, DBPAC aids in data base protection via specified validation and/or conversion routines and coded security classifications.

C. Implementation

This subsystem is used under the IBM TSS system as a user; i.e., it does not involve any alterations to TSS. It is implemented in TSS PL/I except, as anticipated in the contract, for certain TSS functions not supported by PL/I which are accomplished using Assembler Language subroutines. The software is modular for reduction of the implementation time, ease of debugging and maintenance, and "understandability".

It is convenient for the mainline programmers to use this subsystem. (Data base names, and subroutine and macro calls are self-explanatory.)

II. APPROACH

A. Overview

We concluded that the objectives for a data base executive are best satisfied by a combination of:

1. an extension of the PL/I language, called DBPL/I, for data base access,
2. a compilation-time source program processor, DB, and
3. execution-time routines DBPAC, LIST, and # LIST.

In justification of this approach, we quote the TSS PL/I Reference Manual which says: "Since a simple but powerful part of the PL/I language is available for compile-time activity, the generation of text can become elaborate and transformations can be performed. Such transformations might then be considered to be installation-defined extensions of the language." (IBM Form No. C28-2045, Chap. 1)

Mainline PL/I programs have ordinary PL/I input/output statements for non-data base I/O but have DBPL/I statements for all data base I/O such as variable-length VISAM support, variable-length multi-element field access, operations on lists of keys, etc. Such DBPL/I statements logically denote and cause or

influence the transfer of data between the data base and the mainline program. They are similar in meaning and form to ordinary PL/I input/output statements so as to be easily learned and used by mainline PL/I programmers. DBPL/I is independent of all other programs in the system and should not be affected by any reprogramming which may be done later.

The DBPL/I statements are processed at compilation time. This processing is done by the preprocessor stage of the TSS PL/I compiler. A preprocessor procedure (or macro) named DB controls this processing of DBPL/I statements. The DB procedure itself is written using the preprocessor statements available in PL/I. DB analyzes the DBPL/I statements and generates, in their place, ordinary PL/I statements for communication with DBPAC. It generates diagnostic messages if DBPL/I statements have errors or are used inconsistently in a mainline program. If the mainline-DBPAC interface gets respecified, DB can be changed appropriately to generate suitable statements and the mainline programs recompiled without being reprogrammed.

During execution of mainline programs, all data base access is performed by DBPAC, the data base executive. As explained above, mainline programs implicitly call DBPAC, as a subroutine, by their use of DBPL/I statements. For dataplexes or files (including descriptor files), DBPAC can OPEN, CLOSE or ERASE. It can locate a work area for a new record or read an existing record in various ways. It can validate and store or replace field elements. It can extract field elements and transform them for output to the mainline. For retrieval purposes, it can retain lists of keys (e.g., accession numbers) in main storage. A function subroutine, named LIST, can combine such lists in various ways. Another function subroutine, named # LIST, determines the number of elements in any list. DBPAC can read records according to a list and can free lists when they are no longer needed in main storage. The two list functions and most of DBPAC are written in PL/I. Certain interface routines are written in Assembler Language for TSS functions not supported from PL/I; viz., the DDEF command and variable-length VISAM access.

B. Data Base PL/I Language

DBPL/I statements have the same format and syntax as ordinary PL/I statements. They constitute a language for a combination of indexed variable-length record access and multiple variable-length element field access. They must be distinguished for the PL/I preprocessor as follows:

DB ((statements in DBPL/I));

The DBPL/I language is fully specified in the Development Work Book (Section V, Topic B.2) for the guidance of programmers writing mainline programs.

Nearly all statements have a file clause for the dataplex or file name. The file name may be dynamically set by the TITLE option of the OPEN statement. OPEN and CLOSE statements are optional--they need not be used in simple mainlines. The CLOSE statement may have an ERASE option (which can only be executed by a file's owner). An ON ERRORFILE statement is used to establish the label of the user's error routine for a file. The error routine has access to the ONCODE (end of file, key not found, validation failure, etc.), the file name, and the label of the statement following the one that raised the error.

New records are prepared using the LOCATE statement, followed by an appropriate number of PUT statements. When another LOCATE or a READ is executed or the file CLOSEs, the record is implicitly complete and is transmitted to the data base. A READ statement obtains a record from the data base and establishes it as the current record for the file. GET statements are then used to move data to work areas. Any REPUT or PUT statements implicitly cause the record to be rewritten when a READ or LOCATE is executed or the file CLOSEs. READ has variations for reading sequentially forward or backward, for reading by key, and for reading by the next key in a list.

PUT, GET, and REPUT statements implicitly refer to the current record of the file in their FILE clause. PUT FIELD causes one or more field elements to be moved from the mainline, to be validated, to be stored in the record (other fields are shifted to make room, if necessary) for eventual (re)writing and to be cross-referenced in an inverted index file if one exists for the field. PUT RECORD may only be executed by a file's owner and moves a whole record from the mainline (presumably, from a backup file) to the data base output area. GET FIELD causes one or more field elements to be extracted, transformed for output from the data base, and moved to the mainline's work area(s). GET RECORD causes the whole, untransformed record to be moved to the mainline's work area (e.g., for backup). GET SET causes a list of key

fields to be moved from the record to dynamically-allocated main storage and a pointer to it to be set in the mainline. REPUT FIELD causes the last-obtained (or only) element of a field to be replaced by a new value from the mainline. REPUTing a null value from the mainline deletes a field element, left-justifying any remaining elements to close the gap left by the deleted elements. If REPUT FIELD is used to delete the key of the record, the whole record is implicitly deleted, element by element.

Lists of keys (e.g., accession numbers) are implicitly created on inverted index records to cross-reference certain fields. By READing and GET SETting from an inverted index file, a list is brought into and retained in main storage. The LIST function is provided to take two lists and form a new list which is the union or intersection of the two lists, or the first list excluding the second. The new list is retained in dynamically-allocated main storage and its pointer returned to the mainline. The # LIST function determines the number of keys in a list and returns the count to the mainline. A list may be used to control READing of indexed records. The FREE LIST statement is used to de-allocate lists from main storage when they are no longer needed. An ON LISTERROR statement may be used to establish the label of an error routine in the mainline. The error routine may access the ONCODE (main storage exceeded, etc.).

C. Compilation-time Processor

The DB preprocessor procedure's objective is to generate PL/I statements, to accomplish what the DBPL/I statements signify or to generate diagnostic comments for errors that can be detected at compilation time.

Each DBPL/I statement is analyzed individually for correct syntax. If it is correct, PL/I assignment statements and (usually) CALL DBPAC statements are generated in its place and a compilation-time indicator set to note the way the file or dataplex is being used (input, update, etc.). If the individual DBPL/I statement is incorrect, a diagnostic comment is generated and a compilation-time indicator set.

After all DBPL/I statements are preprocessed individually, the compilation-time indicators are examined for each file or dataplex referenced in the program. If the indicators for a given file or dataplex show that it was referenced correctly and consistently throughout the program, then a CALL DBPAC is generated to "automatically" close the file and a DECLARE statement for a Mainline File Control Block (MFCB) is generated. (The assignment and CALL DBPAC statements generated from individual DBPL/I statements refer to fields in the MFCB. PL/I allows fields to be declared at the end of a program.) If the compilation-time indicators for a file or dataplex show any incorrect or inconsistent usage, then a diagnostic comment is generated, but no MFCB declaration. This will prevent the assignment and CALL DBPAC statements, previously generated for the file, from compiling, because they reference undefined qualified names.

Correct ON ERRORFILE and ON LISTERROR DBPL/I statements generate assignments into MFCB's without CALLs to DBPAC. They cause the label of a mainline error routine to be posted for future "asynchronous" reference by DBPAC when an execution-time error is detected. All other DBPL/I statements generate an assignment of an "operation code" (and, for some statements, other fields) to the MFCB, followed by a CALL DBPAC statement with the MFCB as the argument (and, for some statements, an additional argument). GET, PUT, and REPUT FIELD statements are expanded, field by field, by DB so that DBPAC handles only one field per call.

References to the LIST or # LIST functions are not preprocessed by the DB procedure, but the possibility of their presence can be deduced by any use of a GET SET statement and this causes the generation of a statement declaring them as external entry points, returning a pointer or fixed binary value.

D. Execution-time Processor

As has been explained in the preceding paragraphs, DBPAC accepts several different commands from the calling programs and provides the necessary interface with the data base to supply the required data interchange. The commands are the following:

1. OPEN
2. CLOSE
3. LOCATE
4. READ

5. UNLOCK
6. GET
7. PUT
8. REPUT
9. #FIELD
10. FREE LIST
11. LIST
12. # LIST

The following paragraphs discuss (in general) the procedures that are followed with the execution of the commands.

1. OPEN

The process of opening a file results in the establishment of an entry in the MFCB (Mainline File Control Block) and an FCB (File Control Block). Reference can be made to Figure 1, "File Control Block". These elements are more fully described in Section IV, Topic B.2 of the Development Work Book (DWB).

After the MFCB has been posted, the FCB allocated, and the DBPAC switches and element pointers initialized, the address of the proper descriptors is posted in the MFCB.

The security of the data base has been provided on the file and field level. The owner of a dataplex has complete freedom of its use; non-owners are limited by the following restrictions: the files of the dataplex can be opened only as input, and the fields are available only by owner permission and only as input.

DBPAC guarantees the prevention of maintenance within the dataplex when any file is undergoing a reorganization or creation. Conversely, DBPAC sets the proper indicator to guarantee the prevention of reorganization when any given file is undergoing maintenance.

The OPEN command is not required and can be implied by use of the LOCATE command or the READ command (but not READ SEQUENTIAL BACKWARD).

2. CLOSE

The execution of the CLOSE command sets the close switch in the MFCB, writes or rewrites the last record (as required), and closes the file. If the ERASE feature is employed, the file is erased.

The LOCATE, READ, and UNLOCK commands cause the previously worked-on record to be released, written, put, or rewritten (as required). These three commands cause an actual transfer of data either to or from a physical device. They, therefore, require Assembler Language subroutines to handle the variable-length VISAM data sets.

3. LOCATE

The LOCATE command is used for adding records to a file by setting up a null record with the proper key value.

4. READ

a. READ LIST

This command gets the (next) key from a list of keys and indicates to the calling program when none are remaining. It uses this key to read the desired record.

b. READ KEY

Similar to the READ LIST command, READ KEY obtains the desired record from the calling program and establishes it as the current record.

c. READ SEQUENTIAL and READ SEQUENTIAL BACKWARD

The next sequential record (either ascending or descending) is obtained using these commands. The record is then established as the current record.

5. UNLOCK

A record is released for access by other programs when it has been determined that no updating will be done to it.

6. GET

a. GET RECORD

This command gets a record and moves it to the specified area. The owner of the file is the only one permitted use of this command. All records passed from DBPAC to calling programs are variable-length records.

b. GET SET

Through this command, proper storage allocated to the list of anchor file keys and the key's length is posted for control in list functions. It returns a pointer to the calling program to indicate the address of the list.

c. GET FIELD

A field is retrieved from a record and placed in the area specified by the calling program. In the case of fixed fields containing blanks, null fields are returned; in the case of variable-length fields with multiple elements, null fields are returned when all of the elements have been retrieved. Fields are altered by use of the reformat routines specified in the descriptor.

7. PUT and

8. REPUT

a. PUT RECORD

This command obtains a record from the calling program's area and places it in the DBPAC area from which it is written. This command can be used only by the owner of the file and never during maintenance.

b. PUT and REPUT FIELD

The primary function of the PUT and REPUT operations is to put new information into a given position of a record or to have new information replace existing information.

If the calling program sends a null field to replace an existing field (REPUT only), then the following can happen:

a. If the key is nulled, the record is deleted, the proper inverted index fields are deleted, and the proper associated file fields are deleted.

b. If a fixed-length field is nulled, the null value is moved to it after the proper inverted index fields have been deleted.

c. If a single-element variable-length field is nulled, its length indicator is set to a constant of two after the proper inverted index fields have been deleted and after the proper record manipulations have been performed.

d. If a multiple-element (variable- or fixed-length elements) variable-length field (variable number of elements within the field) has one of its elements nulled, its length indicator is algebraically adjusted to represent the proper length after the proper inverted index fields have been deleted and after the proper record manipulations have been performed.

The REPUT command (on a non-nulling field) deletes the old inverted index fields. The commands, PUT and REPUT, on both fixed- and variable-length fields, result in the proper record manipulation which causes the new field to be put or an old field to be reput, effectively, and the new inverted index fields to be created.

Note: These commands are not allowed on inverted indices and REPUTs are not allowed on descriptor files. The field which is being PUT or REPUT is subjected to the validation specified by the descriptor validation routines.

9. #FIELD

This routine is used to determine the number of elements contained in a multiple-element, variable-length field. It can be used for either fixed-length or variable-length elements.

10. FREE LIST

A block of main storage, which had been previously allocated to a list, is freed using this command.

11. LIST

This command performs one of three functions as follows:

a. the 'OR' operation

This operation compares two lists of keys and composes a third list of keys which has, as contents, all of the keys that appeared in each of the two lists. However, keys that appeared in both lists appear only once in the third list.

b. the 'AND' operation

This operation compares two lists of keys and composes a third list of keys which has, as contents, all of the keys that are common to the two lists being compared.

c. the 'NOT' operation

This operation compares two lists of keys and composes a third list of keys which has, as contents, only those keys from the first list which were not present in the second.

Graphically, AND, OR and NOT look as in Figure 2, "Venn Diagrams".

12. #LIST

This command finds the number of anchor file keys (typically, accession numbers) which are in a given list.

There are several different classes of exceptional conditions which might occur during the processing of DBPAC. They are as follows:

a. Expected but infrequent errors, such as end of file and key not found.

b. File reference errors, such as referencing a file that does not exist or trying to OPEN a file for maintenance which is accessible on a read-only basis.

c. Field reference errors, such as referencing a field which is not in existence or which is not available to the user.

d. Command sequence errors in which attempts are made to PUT or REPUT a file which was opened as input.

DBPAC references the MFCB (Mainline File Control Block) to determine if an ON ERROR statement has been executed and an error routine label posted; if it has, DBPAC passes control to the mainline's error routine. Otherwise, the files are CLOSEd and the job is ABENDed.

The errors which can occur and the associated DBPAC error codes can be found in Section III, Topic B.3 of the DWB.

FIGURE I. FILE CONTROL BLOCK

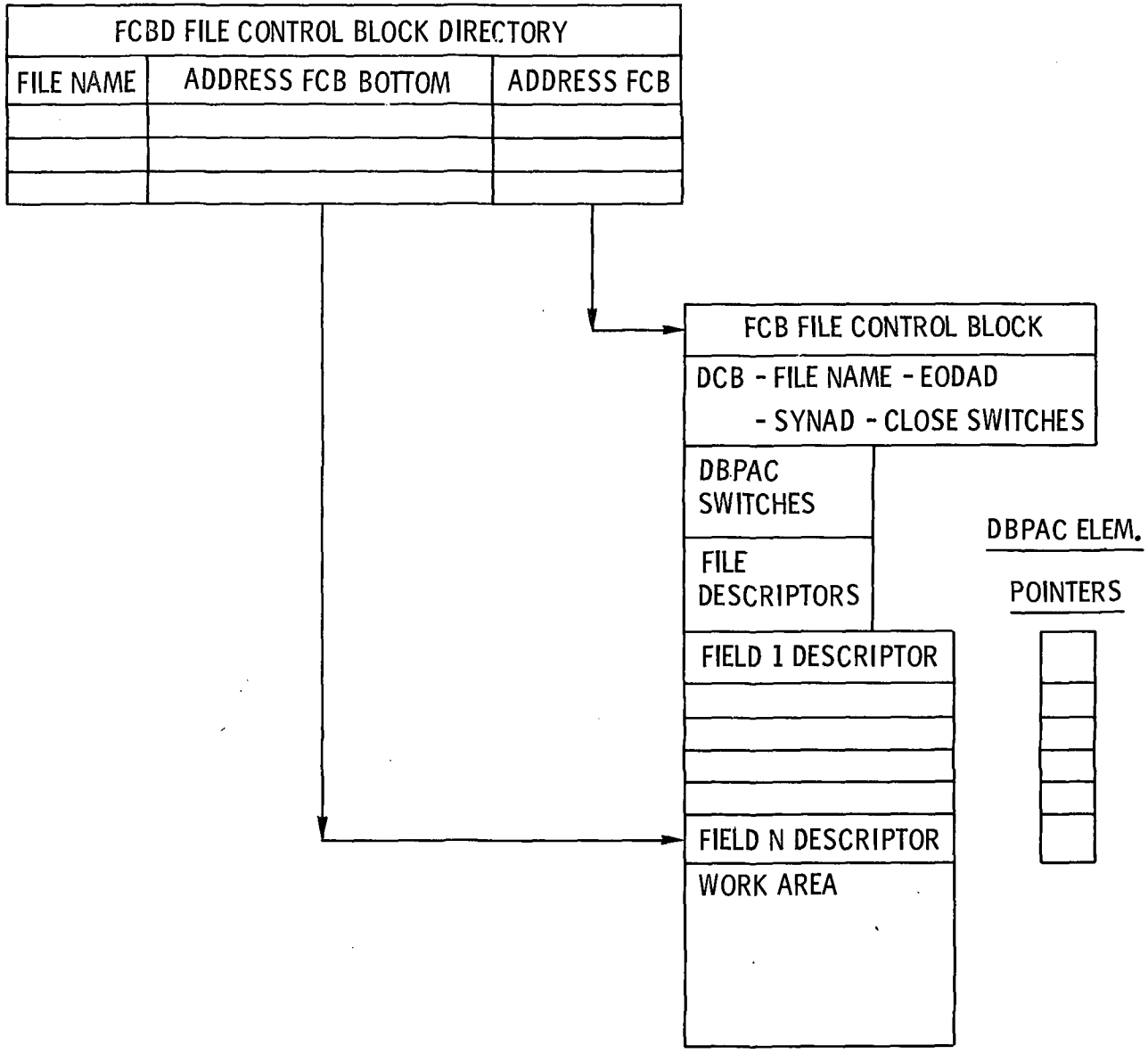
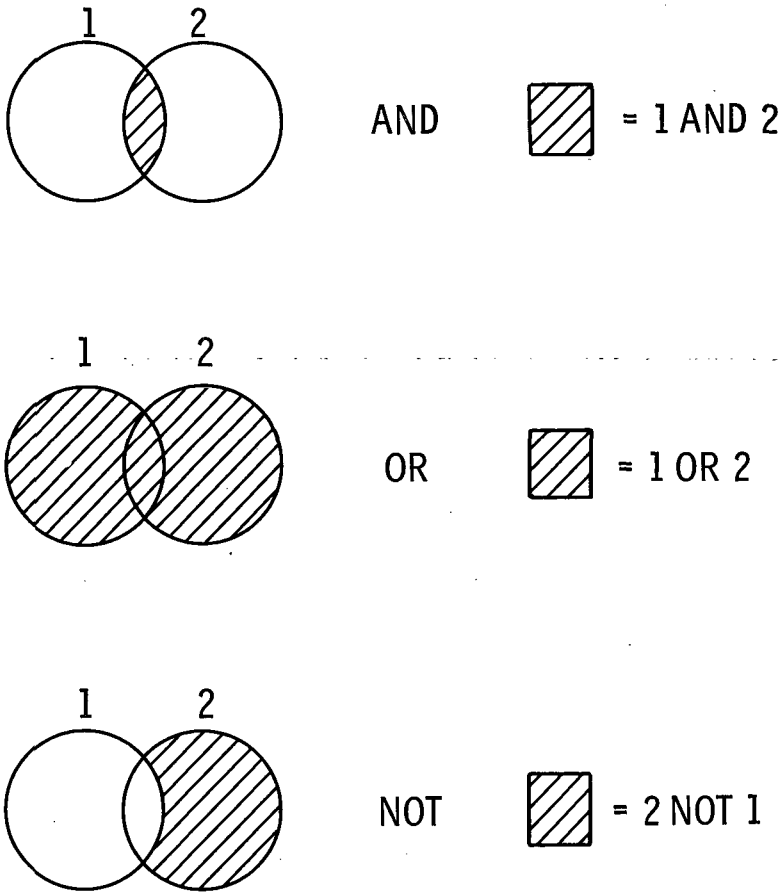


FIGURE 2. VENN DIAGRAMS



TOPIC C.1 - UTILITIES OVERVIEW

I. INTRODUCTION

The utilities subsystem of the NASIS system consists of five major modules:

A. Descriptor Editor
B. Backup
C. Restore
D. Reorganize
E. Purge

These modules, as a group, enable the NASIS user to perform file definition, file dumping, file restoration, file reorganization and file deletion.

II. MODULE OVERVIEWS

A. Descriptor Editor

The Descriptor Editor performs the most important function; that is, the creation and maintenance of the field descriptors. It is designed to be run in an interactive mode where the user is repetitively prompted for the data to be used in defining the descriptors; each of his responses is checked for both syntactic and logical correctness. The user may enter the parameters one at a time or in multiples, if he so desires. The user may execute the program in the batch mode.

B. Backup

The Backup utility performs the function of file dumping. It allows the user to selectively produce an exact copy of any or all of the files that comprise a dataplex. This copy will be produced on magnetic tape in a logical record format.

C. Restore

The Restore utility performs the function of file restoration. It allows the user to use the logical record dumps produced by the Backup utility to reconstruct any or all of the files that comprise a dataplex.

D. Reorganize

The Reorganize utility performs the function of file reorganization. It allows the user to perform a file backup and restore on any or all of the files that comprise a dataplex. This process will restore the efficiency of file access after extensive file maintenance has caused it to be degraded.

E. Purge

The Purge utility performs the function of file deletion. It allows the user to selectively erase any or all of the files that comprise a dataplex.

III. SUMMARY

All of the utilities may be run either conversationally or non-conversationally. They are all parameterized so that the user may simply and easily indicate the function that he wishes performed. With the exception of the Descriptor Editor, the utilities operate interactively only until the required parameters have been successfully entered.

TOPIC D.1 - MAINTENANCE SYSTEMS SPECIFICATION OVERVIEW

I. OVERVIEW

This is an attempt to build a central maintenance package which performs the basic maintenance functions and is modularly expandable in a pre-planned, orderly fashion to include all desired features. A major consideration is providing controls to minimize the chance of individual transactions or entire transaction files escaping inclusion in any maintenance run.

The NASIS maintenance system is composed of the following modules:

A. The Descriptor Editor (RDBEDIT)
B. The Transaction Merge (RDBMERGE)
C. The Load/Create (RDBLOAD)
D. CORRECT (RDBCORR)
E. The Maintenance Mainline (RDBMNTN)

The interrelationships of these various modules are shown in Figure 1a, "Maintenance Subsystem Overview", and Figure 1b, "I/O Block Diagram".

The NASIS system maintenance functions handle the creation of dataplexes and the maintenance (updating) of these dataplexes.

Usually, the NASIS system expects to build dataplexes by subscribing for record dumps of the linear files of other installations. This means that a new dataplex may be added by first creating the dataplex descriptor file using the Descriptor Editor (RDBEDIT) and, secondly, running the record dump tapes through the Load/Create program (RDBLOAD).

The offspring dataplex is maintained and made current with its parent by receiving updated record images of all new and/or changed records and processing these in the update mode through the Load/Create program (RDBLOAD).

The offspring dataplex also is maintained by the transactions created by the CORRECT command. These transactions are processed through the Maintenance Mainline program (RDBMNTN).

The CORRECT command is available to all NASIS users; therefore, there is a transaction dataplex (TRNSCT) under each TSS ID "joined" to NASIS. In order to successfully run the Maintenance Mainline (RDBMNTN) against these transactions, they must be merged together under the dataplex owner's ID. The merge program (RDBMERGE) accomplishes the gathering of all transactions for a particular dataplex. Once these transactions exist under the dataplex owner's jurisdiction, he may use the NASIS COMMAND system to peruse them and the CORRECT command to make changes to them.

The individual modules of the maintenance system are explained in the following paragraphs.

II. MODULE DESCRIPTIONS

A. Descriptor Editor

The Descriptor Editor is a utility program which performs perhaps the most important function of the NASIS system; i.e., the creation and maintenance of the field descriptors. The Descriptior Editor can be executed only by a dataplex owner and may be run either conversationally or non-conversationally.

The editor is invoked by either the CREAT Command, for creating a new set of descriptors or for continuing the creation of an existing set of descriptors, or the UPDAT command, for modifying a set of descriptors for an existing dataplex. The processing of the editor may be divided into three phases: input processing, prompting, and output processing. It should be noted that the editor operates on the descriptor entries in virtual memory; hence, if processing is interrupted before the descriptors are "filed", data will be lost.

The input phase checks for the presence of a dataplex name, prompting if none was entered, and vali-

dates the name. It then attempts to read any old descriptors that may be present, indicating to the user each time an entry is read. If associated files are indicated, their descriptors are also processed.

The prompting phase of the editor allows the user to interactively specify the data from which descriptor entries are to be built. The actions available to the user at this point are the following:

1. list names of descriptors entered thus far,
2. display a formatted description of a particular field entry,
3. add a new field entry,
4. modify an existing field entry,
5. delete an existing field entry,
6. file descriptors and terminate.

The list, display, delete and file functions are trivial and self-explanatory. However, for the remaining functions the user must enter a number of coded or numeric entries to describe the type of field being defined. The user may enter as many of these parameters as he desires, along with the name of the fields. Any parameters not entered will be prompted for one at a time. Each entry is checked for both syntactic and logical corrections.

The output phase is invoked by the "file" command or by an optional response following an attention interrupt. During this phase, the descriptors are sequenced such that the key field is processed first, followed by all bit switches, followed by the fixed fields, followed by the variable fields. As each descriptor is written, the action is indicated to the user. At the end, the user is prompted for the status of the descriptors (complete or incomplete); the program is then terminated.

B. The Merge Program (RDBMERGE)

As indicated in Figure 1b, the results of the CORRECT command are transaction records for a particular dataplex on the TRNSCT dataplexes scattered over several TSS ID's. Therefore, it would be possible to have transactions for a given dataplex existing on the TRNSCT dataplex of every TSS ID joined to the NASIS system.

When the system manager desires to perform maintenance upon an individual dataplex, he wishes to recognize the existence of these pending changes and, after his perusal of them, to include them in the normal maintenance processing.

The merge program is designated to scan the spectrum of transaction dataplexes (TRNSCT) for transactions pertaining to a given dataplex, remove the pertinent transactions from these miscellaneous TRNSCT dataplexes and include them in the system manager's TRNSCT dataplexes.

In this sense, it is a destructive merge, because the transactions are deleted from the source TRNSCT dataplexes when they are included in the master TRNSCT dataplex. However, this eliminates the necessity of purging the TRNSCT dataplexes. Logically, there is no way in which transactions can be lost or not included in any given maintenance run.

Once the transactions have been merged into the master TRNSCT dataplex, it is the system manager's responsibility to peruse the transactions and decide whether they should be included into the maintenance processing. This step is necessary and important to preclude inadvertant information destruction. Very simply then, the merge program brings together all pending transactions into one TRNSCT dataplex for perusal and inclusion into the normal maintenance processing.

C. Load/Create (RDBLOAD)

The purpose of RDBLOAD is to provide a general program for creating dataplexes. As with any general program, the primary obstacle is to resolve the differences particular to each individual implementation. In creating a dataplex, these differences are the nature and format of the input, the name of each field of the output, and the location of the input data element used to create it. In RDBLOAD, the different inputs are handled by creating from them a VISAM data set of variable record length having the same key as the anchor file being created. RDBLOAD recognizes this VISAM data set as a "general input". Since the data base executive uses the descriptor data set to describe the format and nature of each dataplex, the only con-

cerns are, as above, the names of the fields in the anchor data set and the location of the data used to create it. These are different for each particular dataplex, but RDBLOAD resolves this difference by providing the user with an exit in which he, in addition to doing any desired editing, places the input area's address in a pointer array which RDBLOAD uses to create the anchor data set. The fundamental tasks of the user are then to transfer his input to a VISAM data set and code an exit in which he performs any desired data manipulation and a series of equates.

In addition to the primary function of creating the dataplex, the following capabilities are desirable:

1. to test with small volumes of input data;
2. to segment a load by interrupting it and restarting at a specific point;
3. to handle errors in a logical and comprehensive manner;
4. to accumulate, interrogate and dispense statistics;
5. to terminate a create run;
6. to produce an exception-oriented audit trail.

There are several modules to the program: the initial module, the analysis module, the exit/DBPUT module, the exceptional-condition handling module, and the restart handling module. These modules are under control of a mainline module. The program is capable of execution in either a background or an interactive mode; it determines this itself and makes appropriate modifications. The user has the ability to switch from interactive to batch.

The mainline program determines if the program is running in background or interactively, making appropriate adjustments. These include the enabling of appropriate conditions, the disposition of error messages, and the input form of the parameters. Next, it accepts and analyzes those parameters which determine the constraints under which the job runs. These constraints include whether the job is a test or an actual create; threshold values on the number of errors which are tolerated; and the name of the particular dataplex. Having accepted those parameters, all the data sets are opened.

The program reads the descriptor file (for the dataplex). At this point, several possible errors are examined: the possibility that the descriptors could be incomplete; that the user could be attempting to create an anchor file which already exists; and that the user is attempting to restart a load on a non-existent dataplex. If any of these conditions exist, a descriptive error message is logged and the run aborted. When the user decides the load is as desired, he continues that load by restarting from the last record loaded during the final test. Restart from an interrupted create is implemented by RDBLOAD as follows. A search is done to discern the highest key written on the anchor data set; the corresponding record on the input data set is located, thus placing the program logic at precisely the point required to continue from the point of interruption. When the run is not a restart, the arrays for test-monitoring the timings and the testing error actions are initialized and enabled. In either event, the subsequent step is to allocate the pointer arrays for the input data elements and output data names. That having been done, a call is made to the user-coded exit. The parameters passed to this exit are the input record area; the pointer array to the output data names, the pointer array to the input data elements, and a bypass switch, which if turned on, indicates the user wishes to bypass the record. In the exit, the user may do any desired editing of the input; it is his only opportunity. On return from this exit, RDBLOAD writes out the output dataplex. The program then posts appropriate timing and statistical accumulators and processes the next input record, continuing until it reaches a normal end of job or the occurrence of an interruption or an exceptional condition.

Exceptional conditions are handled by a separate module which traps and analyzes each as it occurs. If the condition is a disasterous error, the run is terminated and the user is so advised. If it is a less serious error, the user is advised and the appropriate error accumulators are updated. If it is an actual load, the accumulators are compared to the threshold values; if they are exceeded, the run is terminated. When an "attention" occurs (interactive mode), the action taken depends on whether the run is a test or a load. If it is a load, the user is given the number of records processed, the number bypassed, and the number of errors. If it is a test, the run is terminated.

During a test, the status of each transaction is logged interactively and, hence, the user may terminate if he observes too many errors. An "attention" after displaying the above statistics gives the user three options: to continue interactivly (the default), to terminate, or to enter background mode. Timings for each record-create are taken. During testing, they are used to determine the mean and standard deviation. During actual

load, they are compared to those results. (The purpose of this approach is to prevent the user from running for long periods of time with unproductive results.) When the load time exceeds the mean by a given multiple of the standard deviation, the user interactivly is given the option to terminate or continue. If it is a batch run, the job is terminated. All runs which are terminated can be continued, without disruption, by a subsequent restart.

The last consideration is an orderly end of job handling capability. All terminations enter one routine which logs out the statistics of the run. For tests, this consists of the mean and standard deviation of the time (in seconds) to create each record; for actual runs, it includes the number of input records read, the number bypassed, the number of errors, and the number of records loaded.

D. CORRECT (RDBCORR)

During normal terminal sessions with the NASIS command system, the user is reviewing many details of the information stored on a dataplex. The CORRECT command provides the user with the ability to initiate changes on any incorrect data be may peruse.

The corrections made in this manner during any terminal session result in transactions which are earmarked for updating the particular dataplex being perused.

The CORRECT command format is as follows:

CORRECT KEYVALUE=record-key, FIELDID=field-name, VERIFY=<"YES"/NO>

The CORRECT command provides the NASIS user with an inter-active facility for specifying additions, deletions and changes to the data on an existing dataplex. This facility can be used at any time during a normal terminal session, such that any data errors encountered by the user can be CORRECTed immediately. The "record-key" identifies the identifying field of the record to be modified (usually the accession number). The "field-name" represents the field for which data is to be entered. Absence of this parameter causes a display of the names of all the fields associated with this dataplex. The third parameter is used to indicate whether a user at a typewriter terminal wants a verification display each time he enters a correction, or only upon explicit command. The output from CORRECT is a file of maintenance transactions, one for each sub-command representing a file maintenance action.

There are several sub-commands available within CORRECT. Three of these sub-commands are for program control purposes. They are the following:

FIELDS	which causes a list of the names of all of the fields of the dataplex to be displayed.
VERIFY	which allows the user to change his initial setting of the verification display parameter,
END	which returns the user to normal command mode.

The following sub-commands represent different maintenance actions:

INSERT key	causes a new record to be added to the dataplex, where "key" is the identifying field (accession number) of the record.
ADD data	causes "data" to be added to a null field or element.
REPLACE data	causes existing data strings within the limits specified to be replace new data specified.
DELETE data	causes an element, a range of elements, a field or the entire record to removed from the dataplex.
CANCEL	causes all corrections entered since the last insert or CORRECT to be ignored.

CORRECT allows the user to skip from field to field or record to record without returning to NASIS command mode.

The remaining sub-command is DISPLAY, which is used to present the data contents of the field to the user in a paging mode, where he can scan forward, backward to a particular element or to a particular line. The data displayed will be the data as CORRECTED unless he has made no modifications or entered a CANCEL.

E. Maintenance Mainline (RDBMNTN)

The maintenance mainline program is an independent module which carries out any actual changes necessary to correct, update, or expand the files comprising a dataplex. The specific changes, which can be additions, deletions or replacements, are accepted by maintenance in the form of transactions. The transactions are kept on a dataplex named "TRNSCT" and are created and maintained by the CORRECT command.

The transaction can be applied to the dataplex on a record, field, or element basis. Maintenance produces a file (audit) showing the activity in each dataplex as well as an error report (message) indicating those transactions which could not be applied. Those transactions which are successfully applied to the dataplex are deleted from the transaction file (TRNSCT). Therefore, after the successful completion of a maintenance run, the only transactions remaining on the "TRNSCT" dataplex are those which need correcting. The maintenance mainline acquires the necessary statistics while executing and causes the "STATIC" dataplex (statistics) to be updated (via a call to RDBUPDST).

The maintenance mainline runs only in batch mode. The restart capabilities of the maintenance mainline are inherent due to the deletion of transactions as they are applied and the statistics update after the successful processing of each transaction record.

The maintenance mainline then has external interfaces with modules of the usage statistics. The data set which controls execution of the maintenance mainline also calls RDBPRNTM to cause the maintenance statistics to be printed.

There are several different types of transactions which are acceptable by the maintenance mainline. They are as follows:

a. ADDR is the add-record transaction which indicates that a new record is to be added.

b. ADDE is the add-element transaction which adds fields to the record or elements to a field.

c. DELF is the delete-field transaction which causes fields to be deleted from a record.

d. DELR is the delete-record transaction which causes records tc be deleted from a dataplex.

e. CNGE is the change-element transaction which either replaces a given element with another or deletes it.

f. FLDC is the field-context transaction which permits changes to portions of large fields without the necessity of indicating the entire fields content.

FIGURE Ia. MAINTENANCE SUBSYSTEM OVERVIEW

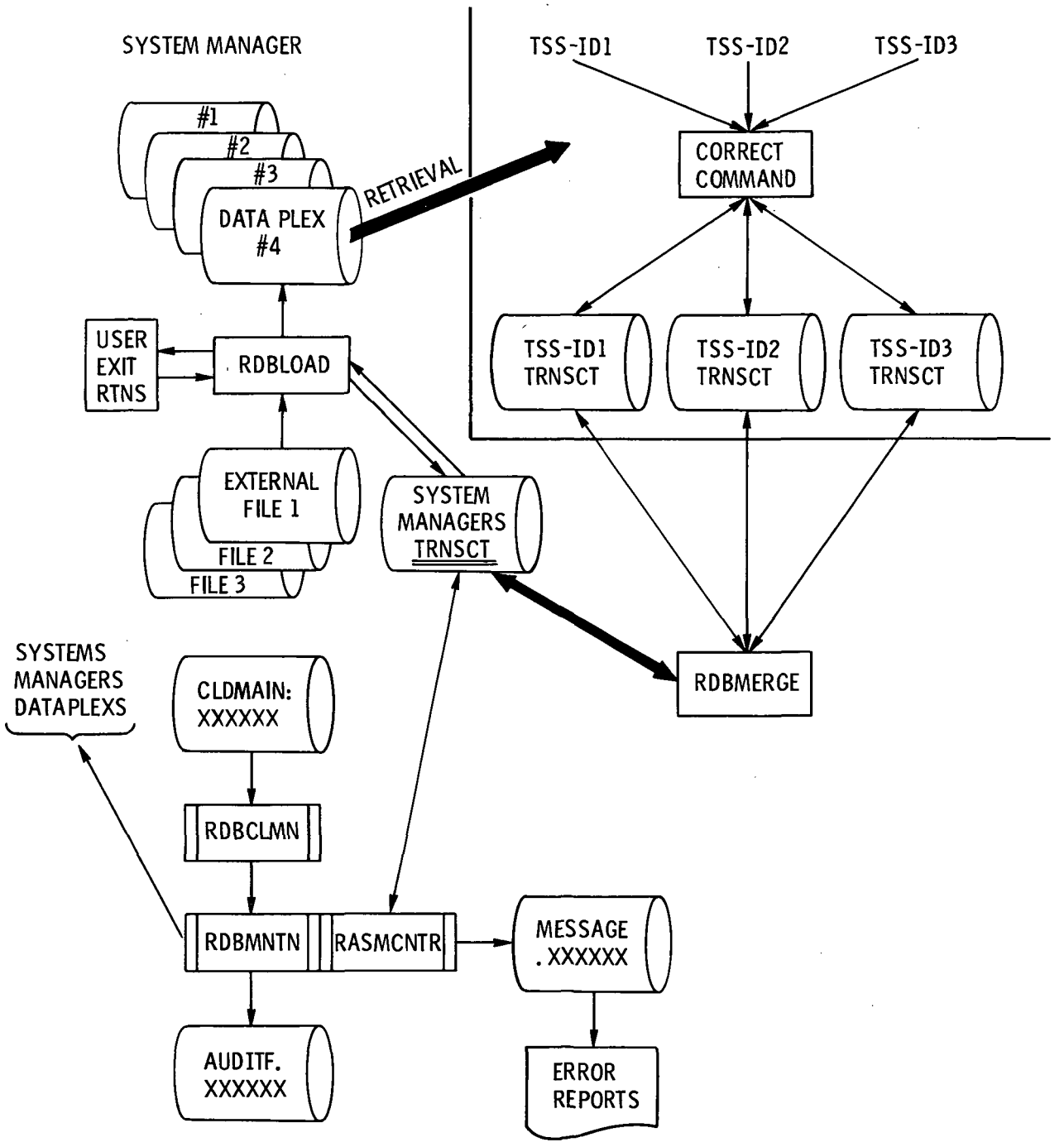
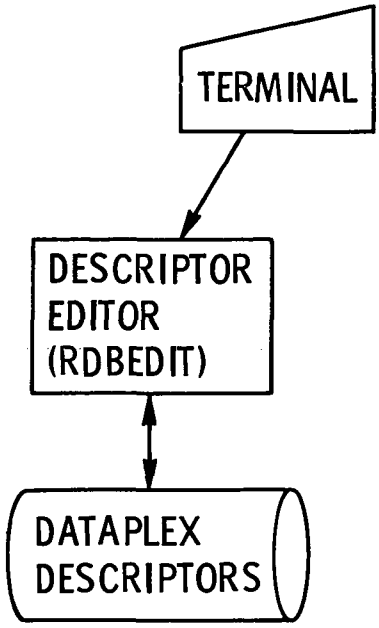


FIGURE Ib.
I/O BLOCK DIAGRAM



TOPIC E.1 - TERMINAL SUPPORT OVERVIEW

I. TERMINAL SUPPORT DEFINITION

A. Purpose

Terminal Support was designed and implemented to provide a coherent means of communicating with a (TSS/360) user terminal from within the PL/I language. A further design objective was to provide a means of communicating with a given CRT terminal (the Computer Communications, Inc. 'CC-30') in the same higher-level language. The last objective was that these two sub-languages be the same.

B. Implementation

The objectives mentioned above were implemented by the creation of four 'facilities' to handle the requisite demands. These are as follows:

1. A PL/I Language Extension (called 'TSPL/I') for writing the source statements necessary to cause I/O to happen at the terminal.

2. A set of programs to serve as a supervisor for the terminal handling programs. (These programs are entitled RTSPRETS and RTSSUP.)

3. A set of programs to format screen images for the CRT and perform the physical I/O to that device.

4. A program to handle the retrieval of explanatory messages for the edification of any user who wishes it.

II. THE LANGUAGE EXTENSION

The PL/I Language Extension designed to serve as the programmer's means of communication with the Terminal Support control programs is entitled 'TSPL/I' and looks to the programmer very much like a normal set of PL/I I/O statements. The facilities provided in the Language Extention (also called the Pre-Processor) are GET, PUT, PROMPT, CONTROL and CLOSE. Also provided are facilities for handling ATTENTION interrupts from the terminal and facilities for handling hardware or software errors from the terminal.

The GET, PUT and CLOSE facilities are much like those used for normal data management and need not be discussed here. The PROMPT facility, however, is unique to the TSS/360 Operating System and hence, a description of it is in order.

TSS/360 supports a file of pre-stored messages for use by the system in its communication with users. (This file is henceforth to be referred to as the 'Message File'.) This concept has several obvious advantages. Each user has the ability to create his own message file, thus tailoring the messages to suit himself. Also, programs containing references to these messages need not have the entire message texts in core; all that is necessary to get to these messages is an 'index' to the message. (These indices are what are called 'Message-Ids' and are in reality keys to an indexed data set which is the Message File.)

The PROMPT facility in Terminal Support is designed to make use of the Message File from within PL/I programs. It behaves more or less like the prompting facilities in TSS/360 with a few restrictions and a few additions.

The CONTROL facility is used for setting-up conditions for Terminal Support such as screen size, line length, and output mode.

The TSPL/I Language Extension User's Guide (Section V, Topic E.2 of the DWB) provides the guidelines for writing TSPL/I statements for use in PL/I programs.

III. THE EXPLAIN FACILITY

This is a facility peculiar to TSS/360 and is closely related to the PROMPT facility discussed above. The EXPLAIN facility allows the retrieval of expanded, explanatory messages dealing with the same subjects as the normal PROMPT messages.

This facility is located within the control-program stucture.

IV. THE SUPERVISORY PROGRAMS

There are two programs in Terminal Support which serve as 'monitors' for the rest of the system. One, RTSPRETS or Pre-Terminal Support merely changes the linkage convention from OS/360 (the PL/I Compiler) to TSS/360. The other program, RTSSUP or the Terminal Support Supervisor, is the main control program for all of Terminal Support.

Basically, RTSSUP is responsible for all linkage between the application programs and the Terminal Support programs, all interrupt/ error handling, all PROMPT message retrievals, all calls on the programs which handle CRT formatting and I/O and all other terminal I/O. If a CRT is not being used in the application, RTSSUP is responsible for everything but the handling of EXPLAIN message requests.

V. THE CRT-HANDLING PROGRAMS

A. Screen Formatting

Within Terminal Support, all the screen images to be sent to the CRT are composed in their entirety in core before they are transmitted to or from the screen. The two programs that perform this function are RTSREAD and RTSWRIT, Terminal Support Read and Write.

These two routines perform, in addition to screen formatting, any text analysis and error reporting (too much text for a screen, too much input for a target string, and so on) necessary.

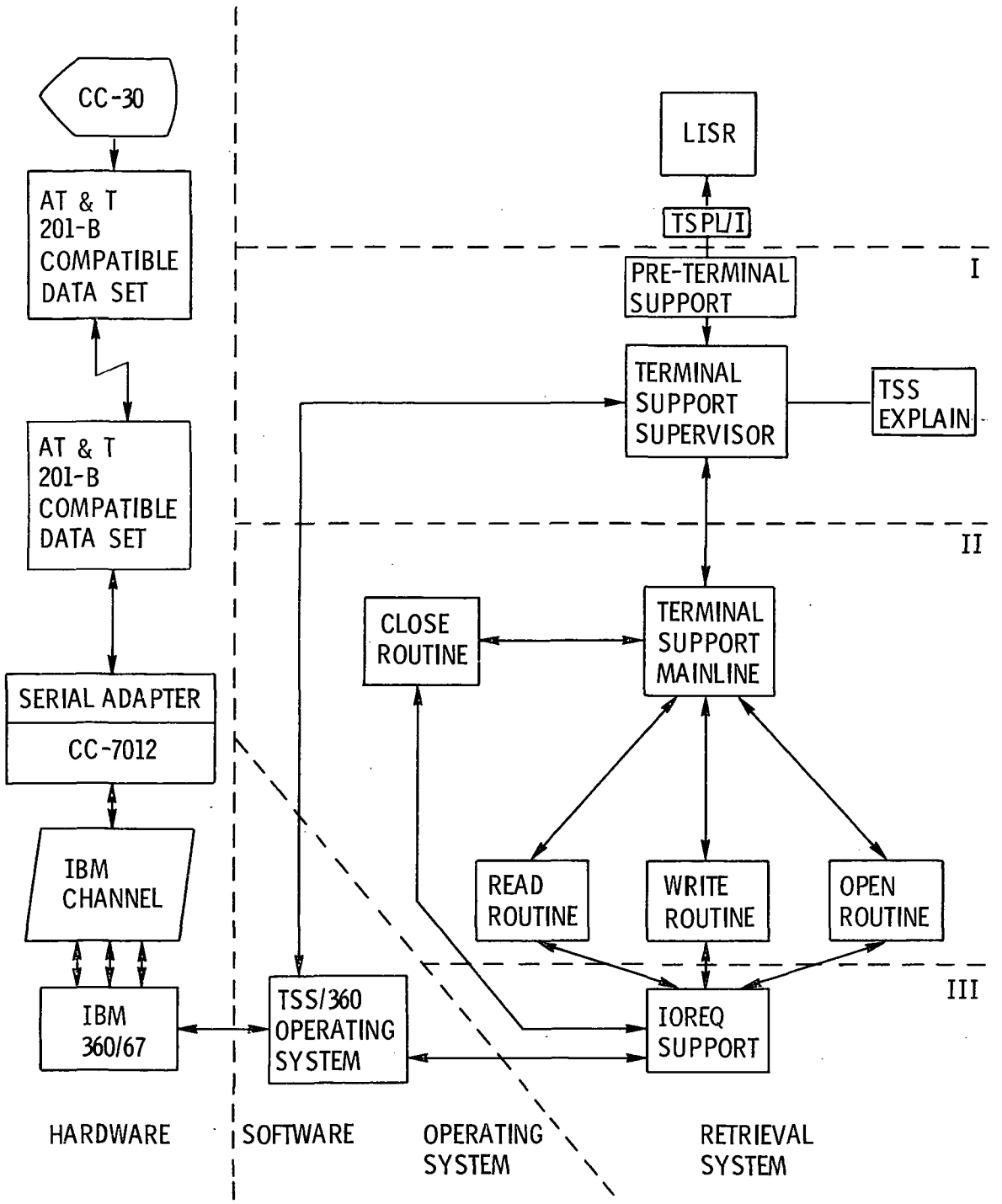
B. The CRT I/O Programs

There are four programs in Terminal Support which are responsible for all I/O done to a CRT. These are RTSMAIN, RTSOPEN, RTSCLOS, and RTSIORQ. The first three of these, Mainline, Open and Close, are responsible for controlling the flow of functions to the terminal, preparing it for I/O and disconnecting it from the application task.

The other program, RTSIORQ or IOREQ Interface, is the program which does all the reading and writing of the CRT, handles error detection and correction (if possible) at the CRT station, also performs data translation to and from the device (EBCDIC-ASCII) and handles the 'OUT' command. This command is used when the user wants a 'hard-copy' of any screen image. RTSIORQ makes use of the CCI printer and issues the I/O commands necessary to 'dump' the screen image to this printer.

FIGURE I.

TERMINAL SUPPORT/RETRIEVAL SYSTEM/360-67/CC-30 INTERFACE OVERVIEW



TOPIC F.1 - RETRIEVAL SUBSYSTEM OVERVIEW

I. OVERVIEW

The entire retrieval subsystem has three components: DIRECTOR, commands, and support routines. The DIRECTOR acts as a controller, funnelling all references of the commands to the proper routine. Data selection and retrieval is accomplished via the commands. Providing utility functions for the commands and the DIRECTOR is a set of support routines which perform initialization, strategy library manipulation, and error diagnostic display. Refer to Figure 1, "Retrieval Subsystem Overview".

A. DIRECTOR

The DIRECTOR controls the retrieval section of the system. It prompts the user for the next retrieval command to be executed and then invokes that command by using the OBEY facility of the IBM 360/67 Time Sharing System (TSS).

Terminal communication consists solely of a prompt of the word ENTER and the entry by the user of the next command to be executed.

B. Retrieval Command Descriptions

The command structure of the retrieval portion of the system consists of seventeen commands which permit initialization of the retrieval system, selection and outputting of records, and preparation of transactions for future maintenance. Each of the commands is described below with all of the parameters indicated and described.

1. BEGIN strategy-name, print99, print98, username,
   useraddress, dataplex, lisrid, security-code

This command is called if the user wants to terminate the current strategy and start anew. The current strategy is saved under the name specified by the "strategy-name". Sets 98 and 99 may or may not be printed, depending on the parameters "print98" and "print99", respectively. The parameters "username", "useraddress", and "dataplex" are used for the entry parameters for starting a new strategy session. If these parameters are defaulted, they remain the same as those in the current session.

2. EXPAND term, inverted-index

This command causes the display of those terms which alphabetically surround the "term" from the cross reference file specified in "inverted-index". The number of terms displayed depends on the number of lines on the terminal screen. To display the next following screen of terms, the user must enter the PAGE command. Looking backward through the index can be achieved by PAGEing backward ("B" parameter).

3. EXSEARCH backsrch=(Y or N), strategy-name

To execute his linear search strategy, the user calls EXSEARCH. All conditionals since the last preceding call to EXSEARCH are performed with the end result being a set of accession numbers in the same manner as the SELECT command. The user can have his search executing conversationally or in background, depending on the value of the "backsrch" parameter. "Strategy-name" indicates the name under which the search strategy should be stored. If the background option is chosen, a prompt is issued for a "strategy-name" if it is not supplied. Also, since a backgrounded execution does not return to the terminal, appropriate PRINT commands, referencing pseudo-sets, must be defined prior to the EXSEARCH command in order to prevent the loss of the results.

During a conversational execution, progress can be determined by using the Attention interrupt feature of the system (TSS). The search status is displayed (records processed, records remaining) for the user's information. Any one of three sub-commands can be entered following the display:

GO which resumes execution

BACKSRCH strategy-name, which forces the execution into the background and stores the strategy under "strategy-name". Prompts are issued if no previous RECORD commands were defined.

CANSRCH which cancels the current search and returns to the point just prior to last EX-SEARCH command.

4. FINISH strategy-name, print99, print98

The FINISH command prompts the user as to the disposition of set-numbers 98 and 99. If "print98" and "print99" are not entered with the command. The user may have them printed on the high-speed printer, after which the sets are erased. If the user does not want them printed, they are erased immediately. The user can specify that he would like his strategy stored under the "strategy-name".

5. FORMAT format, fields

With this command, the user can define his own output formats for his retrieved output. "format" indicates whether the output is a sequential format (one field vertical under another with the field names preceding the data for each new field) or columnar format (typical data processing report with fields spread horizontally across a page or screen. Also, a unique number is given as part of "format" to provide a reference for this new output format (for use with DISPLAY or PRINT). The "fields" parameter provides the names of the fields to be output under this "format". As part of the "fields" parameter, the user can identify the starting output column position for each field and can indicate whether he wants a tally, an arithmetic sum, and/or a statistical mean derived for each field.

Two sub-commands which are provided to the user once he has called FORMAT are TITLE and HEADER. The former sub-command allows definition of report (output) titles; the latter, definition of special row or column headings. Further discussion concerning the FORMAT command can be found in the Report Generator Overview, Section II, Topic F.2 of the DWB.

6. KEEP set-numbers, format, items

KEEP allows the user to accumulate the desired "items" in a special set, set-number 99. The source set and the expected format are specified by "set-number" and "format", respectively. The command is essentially the same as DISPLAY and PRINT except that the output is placed in set-number 99. After accumulating items in set-number 99, the user can, at any time, output its contents using either the DISPLAY or PRINT command.

7. LIMIT set-number, year1, year2,
key1, key2

The LIMIT command is used to place bounds on the records which belong to a set of anchor keys defined by one of the set numbers. "Set-number" identifies the set to which LIMIT is applied. If "set-number" is not specified, then all the subsequently-generated sets are affected. The user may restrict the entries in a set by specifying the "year1" or a range of years ("year1", "year2") in which the documents were published. Restrictions may also be placed on a set of records by specifying a range of keys between which the anchor key must fall. Either or both of the restrictions may be applied.

8. RETRIEVE username, useraddress, dataplex
nasisid, security-code

The RETRIEVE command is used to initiate the retrieval part of the system. This command performs all initialization necessary for the user to use the other retrieval commands. Once RETRIEVE is executed, the user is able to use only the retrieval commands; that is, if the user should enter a utility command, for example, he is issued a diagnostic message, and no action takes place. The parameters "username" and "useraddress" comprise the user's mailing address or location for printed output delivery. The parameter "dataplex" is the name of the file set the user wishes to access.

9. PAGE direction, mode

PAGE is used to change the current screen image to another page within the same display. If "direction" contains a 'B', the last preceding page is displayed. To page forward to the next page, no parameters are needed.

The "mode" parameter is used when displaying multiple anchor records and a record contains more than one screen image. The normal mode (no parameter) is to get the next page of the record until the record is completely displayed. If the user wishes to skip viewing the rest of the current record and start viewing the next record in the set, he enters the 'SKIP' mode with the PAGE command.

The set of records associated with set-number 99, set up by the KEEP command, actually may consist of several disjoint sets of records. When viewing set-number 99, the user can skip from one record to another, using the 'SKIP' mode. It is also possible to jump from one set of records to the first record in the next set by specifying the 'JUMP' mode. For either 'SKIP' or 'JUMP', the 'B' direction indicates backward paging satisfying the particular "mode".

10. RECORD (PRINT) set-number, format, item
(or, alternately) RECORD (PRINT) anchor-key

The following description applies to both the DISPLAY (or REVIEW) and PRINT (or RECORD) commands. The difference is that all output for the DISPLAY command appears at the user's terminal or the user's SYSOUT listing if the task is non-conversational, while the output for the PRINT command appears as a separate listing on the high-speed printer.

The records corresponding to the parameter "set-number" are displayed according to the "format" specified as a parameter. The fixed formats provided by the system allow output of (1) just the anchor keys of the records, (2) the entire record, field by field, and (3) the abstract and anchor key only. Also, a user-definable formatting capability is provided by the "FORMAT" command and can be referenced by DISPLAY and PRINT. For DISPLAY, the user may specify which item is to be displayed first. In PRINT, the user may specify a single number or a range of numbers indicating which items in the set should be printed. If "items" is not specified, the entire set of items is printed (PRINT) or the first item is displayed (DISPLAY). The user has the ability to DISPLAY or PRINT a single record when the anchor key is known.

11. REVIEW (DISPLAY) set-number, format, item
(or, alternately) REVIEW (DISPLAY) anchor-key

12. RERUN strategy-name

This command is used to execute a previously-stored search strategy. The command-strings are retrieved from "strategy-name" and executed individually until the search strategy is exhausted; the user can then continue his strategy.

13. RESTART

The user can use the RESTART command to restore the retrieval system to the point in his strategy that was being executed either when the system abended or when he was forced off the system. The command-strings comprising the current strategy are saved. Each command string is retrieved and re-executed individually until the saved current strategy is exhausted; the user can then continue his strategy.

14. SAVE

The SAVE command is used to save the current screen image appearing in the output area of the display terminal screen. This information is stored in another special set, set-number 98. The user can, at any time, output the contents of set-number 98 using either DISPLAY or PRINT. The SAVE command can be called without interrupting the paging for either DISPLAY, PRINT, or STRATEGY.

15. SEARCH set-number

SEARCH is available for the user who wants to define a linear search strategy, one which accesses non-indexed fields. After executing SELECT as many times as necessary to establish the most restrictive set which can be obtained using indexed fields, the user may wish to further delimit the set using non-indexed fields. "Set-number" identifies the set which will be further delimited.

The SEARCH command then prompts the user for the conditional criteria upon which further delimiting should be based. For each condition, the user specifies the following:

IF field-name op-code val1, val2

where "field-name" indicates the field to be tested, "op-code" defines the relationship (arithmetic, range, or character string context), "val1" is the expected numeric or character string value or lower limit of a range, and "val2" is the upper limit of a range. Any number of conditionals can be defined for the same set-number. The SEARCH command is needed for definition only and does not execute the search or create sets; thus, until the search definition is executed via the EXSEARCH command, the expected results are referenced as pseudo-sets.

16. SELECT expression, inverted-index

The SELECT command outputs the "expression" and the number of anchor keys from the "inverted-index" which satisfy the "expression". A set number is assigned to the list of anchor keys, and the "expression" is entered into the next available line in the current search strategy. The "expression" is a string of characters consisting of any combination, either simple or complex, of the following elements: terms, E-numbers (EXPAND references), and/or set numbers. If there is more than one element in the expression, they must be separated by one of the following binary Boolean operators: "+" (OR), "*" (AND) or "-" (NOT). Any terms specified in the "expression" are resolved from the "inverted-index". Note that the NOT operator can also be used as a unary Boolean operator.

The "expression" can also take the following form:

Xm-Xn/operator

where the X's represent sets of either set numbers or E-numbers, but not mixed: m must be less than n, and the operator must be either a "+" or a "*". This format is equivalent to specifying the following:

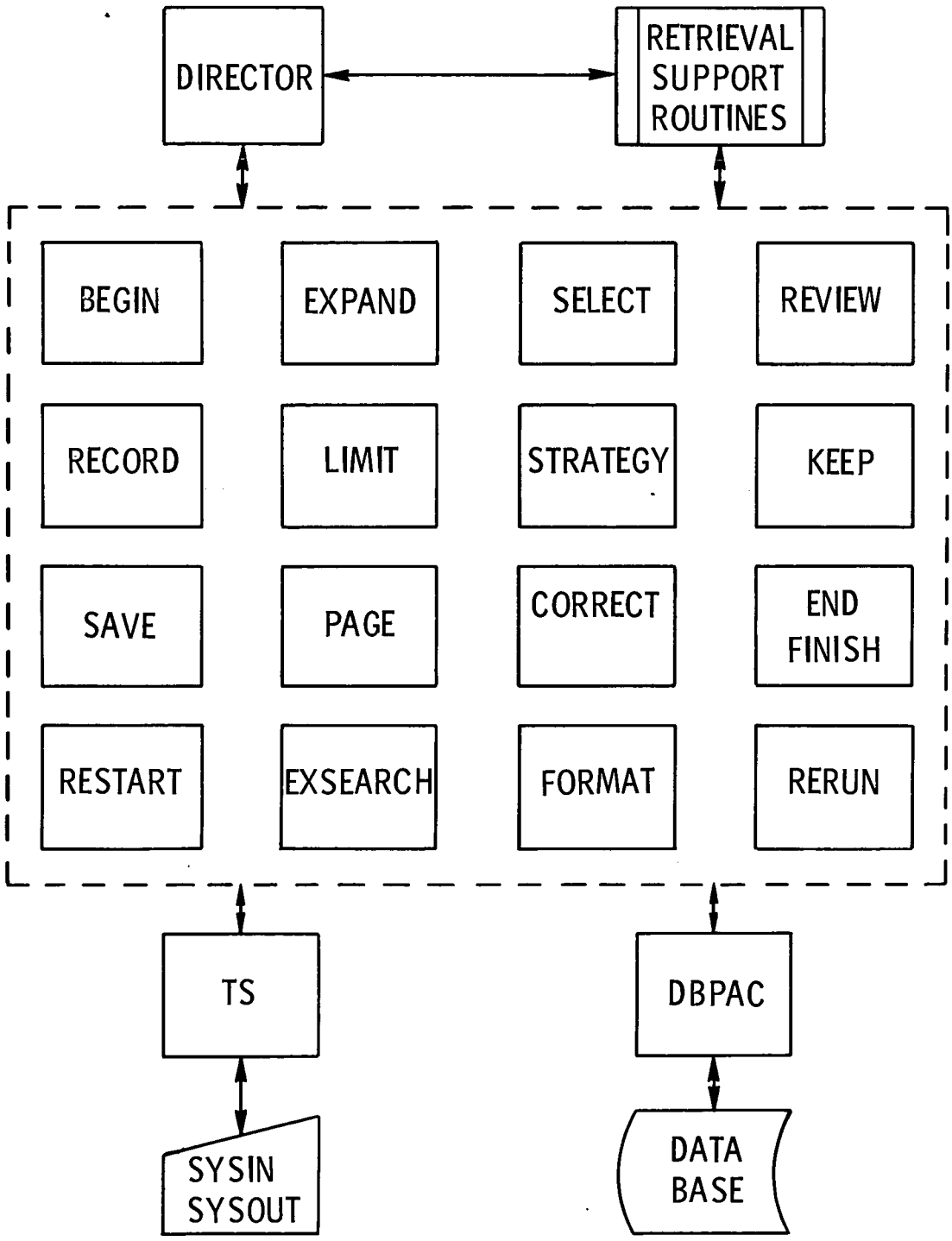
Xm operator X(m+1) operator...X(n-1) operator Xn

NOTE: The SELECT command combines the functions of the RECON commands SELECT and COMBINE. The SELECT search option is described in the Linear Search Overview, Section II, Topic F.3 and in detail in the Linear Search User's Guide, Section V, Topic F.3 of the DWB.

17. STRATEGY strategy-name

Execution of this command causes the search strategy specified in "strategy-name" to be displayed at the user's terminal. If no "strategy-name" is entered, the search strategy currently being executed is accessed; all set-numbers defined in the strategy, their associated item counts, and the SELECT expression from which they were derived are displayed.

FIGURE 1. RETRIEVAL SUBSYSTEM OVERVIEW



TOPIC F.2 - REPORT GENERATOR OVERVIEW

I. INTRODUCTION

The NASIS system provides the capability for the management of voluminous, detailed information. Coincident with this capability are the pressing requirements of a NASIS user to be able to generate an unlimited configuration of information items within a printable report subject to a user-defined format; this requirement is termed a report generator.

The DISPLAY (screen output) and the PRINT (hard-copy output) commands provide a means of formatting a set of information items for output. By modifying these commands to format according to a user-defined format specification, the "outputting mechanism" for a report generator could be achieved within the existing system structure. Hence, a means of format definition, such as a FORMAT command, is a basic system enhancement capable a meeting the unresolvable report generating requirements within the current command structure.

This overview presents and defines the user requirements for a report generation capability of the NASIS system. (The implementation thereof may be accomplished as a command system extension and/or a query language addition.)

II. REQUIREMENTS

A. User-Definable Report Formats

A command-level means of format specification, hereafter called the FORMAT command, allows the NASIS user to stipulate report formats. In general there are two types of formats: columnar and sequential.

A columnar format can be thought of as tabulated data base information where like fields are listed in the same position on a report page, top-to-bottom, and related fields make up a report, line left-to-right. This type of format provides title and column heading capabilities.

A sequential format is a data-base record print-out (one record per page) with the fields therein listed in a sequential manner, top-to-bottom.

With respect to all formats, the user may reference formats by way of some format identifier. This reference may be used with all commands involving formatted data.

B. Screen and Hard-Copy Formats

Due to the various types of and differences between output media, an important requirement is to specify various formats defined for use with a screen (displayable) formatted report and with a printer (hard-copied) formatted report. This requirement provides the user with a dual usage. Besides the output media correlation, the screen format may be utilized to verify interactively the quality of data values destined for eventual printing.

C. Format Strategy Storage

After specifying a detailed or often-used format, the user may need to include the format in his current strategy for future invocation. This requirement is consistent with the current strategy philosophy.

D. Formatting Specifications

In order to facilitate a full data formatting capability to encompass the spectrum of a user's reporting requirements, the following items are at the user's call for inclusion in a reporting format:

1. dataplex field names
2. report column positions
3. field-name column headings
4. report title(s)
5. report pagination

6. arithmetic sum for any field name
7. numeric tally for any field name
8. total of reported items, and
9. the statistical mean (average) per field name.

Nevertheless, the above items have standard defaults upon their exclusion from a report format specification.

E. Inter-Active Format Specification

Although formats are definable in a batch (non-conversational) mode, the interactive facilities of the conversational mode provide the user with a dynamic format specification capability. The user (conversationally) is able to specify (or respecify) formatting keywords and review the effects via a format sample. Even at a later time, the format may be revised in any manner for subsequent review and use.

III. SUMMARY

The FORMAT command provides the user with a means of specifying data items, their page positions, their column heading(s), and their statistical relations along with the overall report title(s) in a very flexible manner. Simultaneously, the format lay-out is reviewable at will during its definition. Many formats are definable and referrable for a variety of uses with other commands. Used with the DISPLAY and PRINT commands, formats may produce a multitude of specialized reports for the same set (or sets) of information. The combined use of the FORMAT command and the format references with other commands provides an unlimited user-definable report generator.

TOPIC F.3 - LINEAR SEARCH OVERVIEW

I. OBJECTIVES

A linear search capability for the NASIS system provides a limited field-oriented non-indexed search facility (LS) as an extension of the NASIS command system. Index files have been defined to provide a more efficient method of accessing large dataplexes. A comprehensive set of capabilities has been incorporated into approximately a dozen simple commands. With these capabilities, it is possible to extract information from any field in the dataplex, providing it has been indexed. Because of the limitations on the amount of direct access storage space, it is not practical to index all fields.

With this in mind, it is evident that for efficiency purposes we have a restraint which takes away some of the flexibility of data retrieval. One way of removing this restraint is to utilize a linear search facility that does not depend on the index files.

LS offers many advantages to the user of the NASIS system. LS allows combinations of individual field search criteria to provide multi-field searches. At the user's discretion, searches are "background-able". Searches are executed on any one existing set of keys or the entire dataplex (file-sets) for each search. Since each search criteria produces a unique set, consecutive searches combined using the SELECT command produce any desirable set of keys. With linear search, it is possible to reduce the number of indices, thus reducing the direct access space requirements of the system, and still provide a complete searching capability for the user.

II. APPROACH

A linear search on a dataplex can be considered as a logical combination of one or more sub-searches which can be defined by a SELECT command search option. The immediate result of a sub-search is a "pseudo-set", or S-number, representing a linear search strategy. Subsequent sub-searches generate additional, unique pseudo-sets. Defined S-numbers, along with the set numbers, may be combined in a SELECT Boolean expression which generates another pseudo-set.

A pseudo-set is unlike a set in that the list of keys it represents is not formed until some later logical point in the user's strategy where he enters an EXSEARCH command. The EXSEARCH command causes the sub-searches to be executed, thus transforming a pseudo-set to a set representing an actual list of keys.

The philosophy incorporated in LS defines a linear search strategy in a unique way. The user generates sets sequentially by combinations of EXPAND and SELECT commands utilizing inverted indices. When he has reached a point where further searching qualifications can not be accomplished via inverted indices, then the LS facility is applied to his one resultant set. The set 0 convention provides the user with the ability to use the LS facility in a situation where no inverted indices can be applied to achieve the desired searching strategy. The above search philosophy thus defines an important LS restriction - the SELECT search command may be applied to only one set for any one LS utilization. That is, only after an EXSEARCH command has been issued can the user apply LS to a different set. Since a set will be generated for each pseudo-set upon termination of the EXSEARCH command, S-numbers will begin again at S1 for subsequent searchers. Note that a pseudo-set may be redefined by the 'SELECT # setnum...' option; the absence of the IF-clause nullifies the pseudo-set.

Use of the EXSEARCH command informs the NASIS system that the user has specified all of his search requests on a set and is now ready to have them executed. If the user requests the background option, he has already entered a PRINT command (involving an S-number) beforehand to obtain the information generated by the LS facility; the user will be prompted for PRINT commands if they do not exist beforehand. The result of the EXSEARCH command is one or more set numbers and their associated lists of accession numbers which the user may reference just as any other set number. Requested PRINTs on S-numbers are performed on corresponding set numbers. During a conversational EXSEARCH processing, the user may wish to see the progress of the search at any time. When an attention interrupt occurs, the EXSEARCH command returns to the user with its current records to be processed. To continue any further in the execution of the linear search, the user then enters one of the following commands:

GO	which resumes the search at the point of execution,
BACKSRCH	which backgrounds the balance of the search, or

CANSRCH which cancels the search in progress, returning the user to the point of his strategy immediately before the last EXSEARCH command.

Should the user desire to background the execution of his search while it is conversationally in progress, he may do so with an attention interrupt, followed by a BACKSRCH command when prompted. This sequence causes the balance of the EXSEARCH processing to be completed in a non-conversational mode. If the user has not specified any PRINT commands previously, he is prompted for them.

This command can only be issued conversationally and only after an EXSEARCH command; otherwise, a diagnostic will be issued.

Should the user desire to terminate the execution of a search while it is conversationally in progress, he may do so via an attention interrupt and, when prompted, the entry of the CANSRCH command. This terminates all of the search requests currently in progress for this task; the NASIS system prompts the user for his next command.

TOPIC G.1 - USAGE STATISTICS OVERVIEW

I. INTRODUCTION

The statistical capability which was decided on for the NASIS system serves multiple purposes. From the maintenance point of view, it gives controls with which that function can be manipulated. The following items are some of the advantages of having maintenance controls:

A. provides the ability to audit the existing dataplexes for correctness,

B. indicates the growth or shrinking of dataplexes for possible device allocations,

C. provides good historical data, and

D. indicates the frequency of change of the various dataplexes in the NASIS system.

The retrieval statistics can be used to show how the usage of the NASIS system is spread over the various dataplexes available, how much (time) each NASIS-user uses the NASIS system and with which dataplexes he is concerned, how to account for all of the computer time which is spent while running the NASIS system, how each user uses the NASIS system in the spread of commands he uses, and how many strategies the user has stored (including their names).

The information available from retrieval statistics can help the system manager determine whether individual people need help in the use of the NASIS system as well as determine how long each individual is running the NASIS system. Since the total computer time per individual NASIS-user is available, it would also be possible to set up an accounting system to allocate the computer time used per each NASIS-user.

Figure 1 depicts the Usage Statistics subsystem.

II. APPROACH

This implementation of the statistics function consists of determining the number of transactions which add, delete, and update anchor records, and determining the number of transactions which imply adds, deletes, and updates to the inverted indices. This provides for record control of the anchor file but does not provide for record control of the inverted indices. These statistics give a good indication of the frequency of change of the dataplex files.

This implementation also consists of user statistics. User statistics provide information regarding the volume, time, and complexity of the user's interaction with the command system and, furthermore, with the inverted files of the dataplex. They are composed primarily of data indicating the number of times a user of the system invokes individual commands on a per file basis, the amount of time a user of the system consumes in both CPU time and connect time, an indication of the length and complexity of the user's strategies, the number of strategies stored, and the total number of terminal sessions.

Within this overview, several different labels may be applied to the same concept. The following list should clarify most of the discrepancies.

Systems Manager Owner-ID: This is a TSS-ID and refers to the owner(s) of the working dataplexes (NSIC, IDEP, COMAT, etc.); to clarify this further, this person is the only one capable of performing creation or maintenance.

TSS-ID: **USER-ID**: In this overview, this refers only to those TSS-IDs (SAFETY**,SAISR1**,etc.) which have been joined to the NASIS system.

NASIS-ID: This is the command system identification given to a user of the command system; its primary use is in the accumulation of statistics.

Security-code: This is a code which is used to allow only certain individuals access to important and restricted fields of a dataplex.

The following paragraphs discuss those requirements which are associated with the implemented statistical capability. These requirements can be classified as those necessary to accumulate and maintain retrieval and maintenance statistics and those necessary to allow retrieval of the statistics.

Maintenance Statistics consist of data accumulated on a per-maintenance-run basis and include the following:

A. The number of transactions per fileplex (anchor and all associated fields), classified as transactions representing new records, updates, or deletions.

B. The number of transactions per the XPLEX (all inverted files of a dataplex), classified as transactions representing new records, updates or deletions.

Inverted files consist of keys which represent non-null dataplex anchor fields; therefore, transactions which affect the fileplex may also affect the XPLEX.

Retrieval statistics consist of the connect time, the CPU time, the total strategy length, the number of strategies currently stored (limited to 3), the data set names of the stored strategies, the total number of terminal sessions and the date of the first and last terminal sessions; all kept by the NASIS-ID of the command system user. Furthermore, the retrieval statistics include the number of EXPANDs, SELECTs, SEARCHs, and CORRECTs per session for the last twelve sessions and accumulated prior to that, and the date of the last thirteen sessions; all kept by NASIS-ID/OWNER-ID/dataplex-file.

The retrieval of statistics consists of two reports. The first report contains the following information by NASIS-ID:

A. The activity of the XPLEX (inverted indices); i.e., the average number of SELECTs and EXPANDs per session.

B. The average length of a strategy per session.

C. The total connect time and the total CPU time.

D. The number of strategies currently stored, a list of these strategies (names), and when they were last used.

E. The average number of times the CORRECT command was invoked per session.

F. The total number of terminal sessions.

The second report contains the following information by dataplex:

A. The total number of transactions per maintenance run.

B. Total records currently in the dataplex anchor file.

C. The average percentage of the file affected by the maintenance runs.

D. The frequency of maintenance runs (date).

E. Statistics per maintenance run of add, delete and update transactions.

The first report is provided by the RDBPRNTR module and the second report is provided by the RDBPRNTM module.

The statistics accumulated are all kept on a dataplex. Each TSS-ID joined to the NASIS system has its own statistics dataplex, and it can be shared with other TSS-IDs just as any other dataplex. The dataplex name is STATIC.

All joined TSS-IDs of the NASIS system must know all the other joined TSS-IDs. This is true only of the system managers (i.e., those who own their own dataplexes) in order for them to perform maintenance on their

dataplexes; therefore, a data set is created under a single TSS-ID (SAFETY**). This data set contains the names of all joined TSS-IDs. This data set will be shared read/write access with all other joined TSS-IDs with the name NASIS.JOINIDS.

In order to keep statistics per each NASIS-ID of the command system, it is necessary for each user (person or group) to have his own NASIS-ID. This individually-assigned NASIS-ID is not a TSS-ID but, instead, is a password. For example, an individual could be executing the command system under the TSS-ID of SAFETY**, yet his NASIS-ID could be PRITCHRD.

This concept requires a program to allow the system manager to assign NASIS-IDs and the various dataplexes allowed to each of these NASIS-IDs; this program is called RDBJOIN. These items are all posted to the data set NASIS.NASISIDS.

All of the maintenance statistics are automatically updated with the Load/Create program (RDBLOAD) and the Maintenance Mainline program (RDBMNTN). If the dataplex owner wishes to modify certain data pertaining to the maintenance statistics, he may use the CORRECT command.

RDBACCUM is used to accumulate the maintenance statistics on those dataplexs which have already been loaded or are loaded outside of the NASIS system.

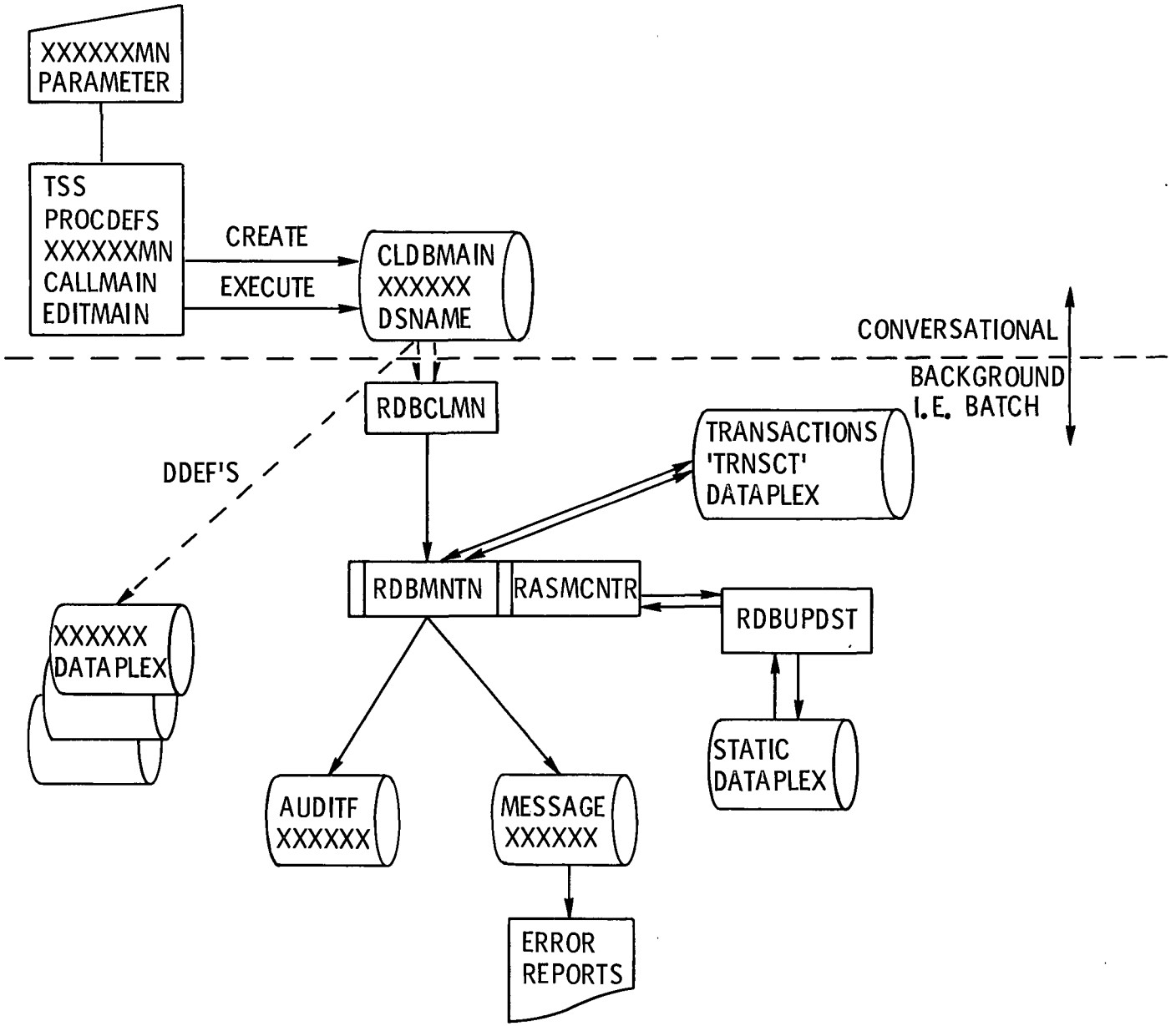
Hand-in-hand with placing the statistics information on a dataplex goes the creation of all transactions as a dataplex. Each USER-ID has a dataplex, TRNSCT, which contains all of the transactions created under that given joined TSS-ID. The CORRECT command could be used, in effect, to perform on-line maintenance to not only the STATIC dataplex but the TRNSCT dataplex as well.

The Load/Create program handles the TRNSCT dataplex in an update mode. A switch in the fields of the TRNSCT dataplex indicates when the transaction record has modified a given dataplex, or the record itself is deleted from the dataplex. This allows the Maintenance Mainline program to have restart capabilities based on the last record processed. The statistics on a per-dataplex basis in conjunction with the Maintenance Mainline are used to perform control functions on a dataplex. The original number of records on the dataplex, plus the adds and minus the deletes, is the current number of records. If any discrepancies occur, they are then detected and eliminated.

The creation of the transaction dataplex has, as a requirement, the creation of a program which converts all of the existing transaction files to a TRNSCT dataplex. The primary purpose of this program is to perform that conversion.

All the retrieval statistics are automatically updated within the modules of the NASIS command system. If required, at maintenance time a "snapshot" of the statistics is printed. This is done by the RDBCHKPT module. If the dataplex owner wishes to modify certain data pertaining to the retrieval statistics, he may use the CORRECT command.

FIGURE I. USAGE STATISTICS OVERVIEW



TOPIC H.1 - MT/T MONITOR OVERVIEW

I. OBJECTIVES

A. Problem

The NASIS system as currently implemented has several serious drawbacks. Due to the implementation of PL/I under TSS/360, the NASIS system is such that each user must have his own copy of the entire system. Needless to say, this results in a serious constriction of the system's paging facilities as soon as a few users begin running NASIS. Also, administrative control over the system is limited as there may be several copies of the system running.

B. Solution

There is only one practical way to solve all the above-mentioned constraints: to run NASIS as an MT/T application. An MT/T (Multi-Terminals/Task) application is a single TSS/360 task having the ability to service as many terminals as necessary. In effect, NASIS is an operating system. (MT/T was originally conceived as the solution to this same problem but relative to "computing" tasks like desk calculators, in-core editors and so on. The most spectacular MT/T application to date is the use of the facility to drive APL/360, developed at the Carnegie-Mellon University (under contract) by A. L. Vareha and J. M. McCrossin.)

II. APPROACH AND DESIGN

A. Internal Features

For use with NASIS, the MT/T Monitor "sits on top of" the remainder of the system; that is, it is the first-level interface between both TSS/360 and NASIS and between the User and NASIS. This Monitor performs operating-system functions for NASIS and terminal-processing functions for the user.

The operating-system functions include multi-programming, time-slicing, terminal-keeping, user-to-user communications, all requests for reading/writing the terminal and scheduling of sub-tasks.

Basically the flow through the Monitor is as follows. After an entry to the Monitor is made for the purpose of I/O to a terminal, the concerned section of the Monitor processes the request as far as possible (that is to the point of actually issuing the I/O request to the terminal) and then calls a section of the Monitor called MTTFINDQ. This section, the Queue Finder, is responsible for all the multi-programming work within the Monitor. It searches all the Queues that the monitor used for work to be done, takes care of all scheduling and timeing, and takes care of all task dispatching. After the original I/O request is finished, the Queue-Finder returns to the section to complete processing of the I/O completion. Hence, basic flow through the Monitor is I/O request to multi-program processing to I/O completion. Of course there is more to it than that: task dispatching, time slicing, queue manipulation, operator-user communication, user-user communication; but all this fits neatly into the scheme described.

B. External Features

The most obvious external features are the solution of the problems described in Section I. Otherwise, running the system only in one "place" has the advantages that the operator can talk to everybody with relative ease (he doesn't have to send messages to several TSS tasks); the operator can schedule the system to fit other needs (he can shut the system down at any time and with any amount of time-warning); and he can monitor the usage of the system at any moment with no trouble. Furthermore, password/security processing is simplified as the Monitor is the thing that processes all passwords and security coding.

C. Specifications

This Monitor is a single TSS/360 program written in TSS/360 machine code aided by the TSS/360 system macros and the MT/T queue manipulating macros. The monitor runs in "privileged" mode and under a userid having the MT/T option (i.e., Authority "O" and Privilege Class "T"). All I/O performed by the Monitor is processed in the "polling mode"; asynchronous interrupts are processed by the (TSS/360) system

and return indications sent back to the Monitor. (In the current NASIS system, these are handled truly aschronously; the new method is much simpler and less prone to system complications.)

NASIS OVERVIEW'S GLOSSARY

"Accession Number" - A NASIS-oriented term meaning a key which servies to uniquely identify a record.

"Access Method" - A software link which aids the user in transferring data into or out of memory from a secondary storage device.

"Algorithm" - A sequence of instructions which perform a desired task in a definite deterministic manner.

"Anchor File" - A NASIS-oriented term to indicate the file containing "anchor" information i.e., pertinent, often-accessed information. This information is usually stored on a fast-access medium.

"Associated File" - A NASIS-oriented term indicating one or more associated files which may exist in parallel with the anchor file. The records of each of these files are extensions of the anchor file and have the same key values. These records usually consist of long infrequently accessed data. These data may be stored on slower, more economical devices and are mutually exclusive of the data in the anchor file.

"ASRDI" - Aerospace Safety Research and Data Institute.

"Asychronous" - That which is not synchronized with respect to any other event; an event whose occurrence is not synchronized with program execution and thus may occur at any time.

"Background" - Nonreal-time, nonconversational lower priority tasks run on a time-sharing or multiprogramming system whenever high-priority or quick-response programs are inactive.

"Backup" - Auxiliary storage (tape, disk, cards) containing the most recent copy of a system to be used in case the current system is accidentally destroyed.

"Batch Processing" - A technique that loads and activates a single job which then processes several unrelated jobs sequentially, without having to be reloaded or reinitialized for each job.

"Boolean" - Pertaining to the algebra developed by George S. Boole (1815-1864). It is used today in formal logic systems to express relations between sets. Its main logical operators (AND, OR, NOT) are equivalent to the ordinary algebraic operations (*, +, -).

"Byte" - A string of binary digits operated on as a unit. It is the smallest division of memory which can be accessed as a unit.

"Compiler" - A program to translate a higher level language (e.g., PL/I, COBOL, FORTRAN) into machine language.

"Compiler-Oriented Language" - A language, normally computer independent, designed to facilitate the accurate description of procedures, algorithms, or routines belonging to a certain set of procedures.

"Connect Time" - The total time a terminal is attached to the computer during a terminal session i.e., the total elapsed time between LOGON and LOGOFF.

"Conversational" - The interaction between a user and a time-sharing system, via a terminal device. The user can enter his task to the system and then direct the system as to the manner of processing the task.

"CPU Time" - The total elapsed time the Central Processing Unit was dedicated to running a user's task.

"CRT" - Cathode Ray Tube - a T.V.like device most commonly used in graphics applications.

"Cross Reference File" - A file associated with some field in a NASIS anchor record. Its records consist of field values of the anchor record field and key values of the anchor records having this specific field value.

"Data Base" - A collection of machine-readable files so structured as to contain in an efficient and comprehensive

manner the total information content of a particular application.

"Dataplex" - A NASIS-oriented term indicating a data structure consisting of a FILEPLEX and an XPLEX.

"Dataset (Data Set)" - A named collection of logically related data records stored upon some machine-readable medium.

"Direct Access" - A type of storage medium that allows information to be accessed by positioning the medium or accessing mechanism directly at the location of the information required.

"Descriptor File" - A NASIS-oriented name indicating a file whose records contain a set of values which define the attributes of data fields of the files within a DATAPLEX.

"Dynamic Allocation" - Providing storage to a program based upon the actual instantaneous need for storage space, rather than its anticipated or predicted demand.

"Element" - A NASIS-oriented term indicating a logical data group within a field.

"Entry Point" - Any location in a program to which control is passed from another program.

"Executive" - A routine that controls the execution of other routines. This simplifies the operator intervention involved in running a task and insures a more effieient operation.

"External Entry Point" - An entry point referenced by this program but defined in another program.

"FCB - File Control Block" - This is a data-base-executive created control block. It is an array of data control blocks, work areas, and, possibly, null records required to handle the field contained in the MFCB.

"Field" - A specific area used for a particular category of data. In the NASIS System "field" represents a logical data group in a record.

"File" - A collection of related records treated as a unit. A physical file refers to the actual physical device on which the data are stored. A logical file is defined in terms of the information it contains independent of the medium on which the data are stored. Thus, a logical file might exist on two different physical devices.

"FILEPLEX" - The NASIS-oriented name given to the anchor file and its associated file(s).

"Global Variable" - A variable whose value is defined throughout a program as opposed to a local variable whose value is known only within the block in which it is defined.

"Indicator" - See "Switch".

"Information Retrieval" - The technique and process of accumulating, classifying, cataloging, storing, and searching large amounts of data, and reproducing or displaying the required information contained in the data on demand.

"Information Storage" - A group of storage devices, not necessarily the same, used to hold information.

"Interactive Mode" - The executing of a task conversationally.

"Interface" - The place at which two different systems or sub-systems meet and interact with each other.

"Internal Form" - The actual binary representation of a program or data within a computer.

"Inverted Index" - A NASIS-oriented term referring to a cross referenced file. It is an index to the values of an information field (either in the anchor file records or in an associated file's records). Each of the records has a value of the index field as its key value and the list of anchor file keys as data.

"Key" - A data item that serves to uniquely identify a record. The key may be imbedded within the record, or it may immediately precede the record, or it may be separated from the record and contain the address of the rest of the record.

"Linear File" - A file in which the data are stored sequentially.

"List" - A group of items in the system waiting for the attention of the processor. The number of items in the list may vary from time to time.

"List Processor" - A program in which list structures are used to organize memory.

"Lock Out" - The ability within a system to reserve any resource for exclusive use. Thus, if some resource is being updated, the user doing the updating has the ability to exclude all other users from this resource until he is through using it.

"Macro Instruction" - One line of program code which generates a program routine rather than one program instruction.

"Mainline" - A NASIS-oriented term referring to any control routine that functionally performs the various actions of the system by directing calls to the executive routines for data base I/O and performing all internal manipulations on the data.

"MFCB - Mainline File Control Block" - A NASIS-orinted control block defined in DBPL/I. It is a direction table which is used by a mainline program to successfully complete a necessary task.

"Module" - A functional programming unit having known properties. Usually programming modues are so constructed so as to be used interchangeably with a number of different prograMS. Thus, the problem of maintaining a large system is reduced.

"Monitor" - A collection of program modules that control and supervise the usage of a system.

"Multiprogramming" - A technique by which a computing system can be used to execute two or more unrelated programs concurrently.

"MT/T - Multi-Terminal/Task" - This is a single task running under TSS/360 having the ability to service as many terminals as necessary.

"On-Line" - Referring to a device which is capable of interfacing with the computer. The device operates under the control of the computer and is used in cooperation with the computer to accomplish a task.

"Opcode" - A symbolic or numeric code indicating to a processor a specific operation to be done.

"Operation" - A process or a procedure that obtains a unique result from any permissable combination of operands, or the sequence of actions resulting from the execution of one digital instruction.

"Operating System" - A set of programs and routines which guide the computer in the performance of its tasks, assists both programs and programmers with supporting functions, and increases the usefulness of the computing hardware.

"Page" - A standard quantity of main-memory capacity usually 512 to 4096 bytes. Pages are used in allocating memory and partitioning programs into control sections.

"Parameter" - A quantity that is assigned a temporary value in order to modify control or influence a process or a procedure.

"PL/I" - A multipurpose programming language developed by IBM for the System/360 which can be used for commercial and scientific purposes.

The following define some of the terms used in the PL/I language. They are not meant to be all inclusive.

a. BUILT-IN function - a function which has already been described in a higher level language.

b. CHARACTER-STRING - A PL/I data item which can be constant or variable and whose "value" is a string of characters.

c. DECLARE statement - A PL/I statement which lists the names of all variables used within the program and their attributes.

d. DO group - A group of statements in PL/I beginning with the DO statement and ending with the END statement. A DO-group behaves much like a single statement within the PL/I syntax.

e. dynamically encompassing block - The outermost block within which the value of a specified variable is known.

f. END - This statement indicates the end of a block.

g. external procedure - A procedure entirely defined outside the procedure from which reference is made.

h. FIXED BINARY - An attribute of a variable declaring that its internal representation will be binary and its length fixed.

i. GO TO statement - A transfer of control statement.

j. LENGTH - A built in function used in connection with character strings. It will return an integer value which is the number of characters in the string.

k. Pointer variable - A variable whose value is the location of some variable.

l. RETURN - A statement which causes control to be returned to the calling procedure and from there the program continues in sequence.

"Post" - To record the occurrence of an event for later interrogation by a procedure. The action of the procedure will depend upon the status of the event.

"Preprocessor" - A program which acts upon data prior to sending it to a processor.

"Print98" - A NASIS data set used if the user wished to SAVE the current screen image appearing in the output area of the display terminal.

"Print99" - An in core table used to KEEP information normally processed by the REVIEW/RECORD commands. After a terminal session this table of commands can be REVIEWed or RECORDed totally or selectively.

"Procedure" - A sequence of actions which collectively accomplish some task.

"Processor" - A program which transforms some specified input stream into some desired output stream. Processors are normally referred to as assemblers or compilers.

"Qualified Name" - A name further identified by associating it with additional names. Usually names are qualified in an hierarchical manner. This allows for an explicit reference to a resource or a more general reference to a group of resources.

"Queue" - A group of items in the system waiting to be acted upon by some processor.

"Read-Only" - An attribute of some medium or procedure indicating that the contents of the medium or procedure cannot be altered during use.

"Record" - A group of related data items treated as a unit.

"Region Dataset" - The region data set is a special form of a VISAM data set. Records in region datasets are indexed by two concatenated fields; region names and line numbers. Region names arranged alphabetically, divide the dataset into regions; line numbers index the lines within regions.

"Response Time" - The time it takes for the system to react to a given input. More generally, the response time could be defined as the interval between an event and the system's response to the event.

"Restart" - The process of beginning a program anew at some point determined during a prior execution of the program. Usually some catastrophic event has occurred during a previous run and the programmer wants to continue running the program from the last known valid point.

"Retrieval" - See "Information Retrieval".

"Routine" - A sequence of machine instructions that direct the execution of a well-defined function.

"Search" - The process of seeking a desired item or condition as a set of related or similar items or conditions.

"Self-Organizing" - The capability of a system to arrange its own internal structure.

"Sequential Access" - A process that involves reading or writing data serially and, by extension, a data medium that must be read serially.

"Set" - A list of anchor file keys which are directly coupled to the expression in the most recent SELECT command. (i.e., The SELECT command searches the inverted index file specified for the expression specified. All hits are stored in a special list called a set.)

"Set-number" - A number assigned to a set. Currently this number can range from 1 to 97.

"Source Program" - The form of a program just as the programmer has written it. This usually exists on coding forms or on some machine readable medium.

"Storage" - Any device that can accept, retain, and read back data one or more times. Usually synonymous with memory.

"Strategy" - A NASIS-oriented term indicating a sequence of commands used during a terminal session. This sequence can be saved. Thus, a person has a hard copy of the logic he used during the terminal session. If desired, this sequence can be executed automatically at a later date and the user can continue from that point.

"Strategy-name" - The name given to the data set containing the user's strategy.

"String" - A set of consecutive, adjacent items of similar type.

"Switch" - A hardware or programmed device for indicating that one of several states or conditions have been chosen, or to interchange or exchange two data items.

"Syntax" - A set of rules needed to construct valid expressions or sentences in a language.

"Syntax Analysis" - The process of determining whether or not the syntax of a given string is valid.

"Table Driven" - A process whereby the validity of the input is determined by a processor whose actions are directed by the contents of a table.

"Task" - A set of instructions, data, and control information capable of being executed by the computer in order to accomplish some purpose.

"TCB - Terminal Control Block" - A NASIS-oriented term used in TSPL/I. This control block provides for control interface between the user's program and TSPL/I.

"Thesaurus File" - A separate dataplex under the NASIS system. It contains synonyms of key words, more encompassing keywords, or more specific key words. It is to aid the user in preparing his strategy.

"Time-Sharing" - A method of using a computing system that allows a number of users to execute programs concurrently and to interact with them during execution.

"Time Slice" - The time allotted to a user during the complete execution-time cycle for all users of a time-sharing system.

"Trap" - An automatic transfer of control to a known location occurring when a specified condition is detected by hardware. A trap is to be distinguished from an interrupt in that a trap is caused only by the CPU, the program, or some internal event.

"TSS/360 - Time Sharing System/360" - An operating system for large-scale 360 configurations having time-sharing features.

"Unary Operation" - An operation in which only a single operand is required in order to give a unique result.

"Universal-record-format" - The generalized record format used in the NASIS system. It consists of a record all of whose variable fields follow its fixed fields. This allows high-order addressing of fixed fields and relative-position addressing of variable fields. The net result is space savings.

"Variable Length Record" - A data or file format that does not specify exactly the size of each individual record, but allows each record to be exactly as long as needed up to some specified fixed maximum.

"Virtual Memory" - A programming technique used in operating systems to make the actual physical memory appear (virtual) to be larger than it actually is. This is done by an elaborate mapping technique which causes currently unneeded portions (pages) of the user's program to be written to an auxillary storage device (usually a drum). These pages will be recalled when the user's program has need of them. This whole technique is transparent to the user and he thus has the sensation of having the entire "virtual" machine to himself.

"VISAM - Virtual Indexed Sequential Access Method" - An IBM-supplied access method which organizes records into an ascending collating sequence, based upon a data key associated with each record.

"Wait" - A condition which indicates that a task cannot continue until a specified event or combination of events occurs.

"XPLEX" - A NASIS-oriented term indicating a group of cross reference files associated with a specified FILEPLEX.

NASA-Langley, 1972 — 34 E-6581

NATIONAL AERONAUTICS AND SPACE ADMISTRATION
WASHINGTON, D.C. 20546

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE $300

FIRST CLASS MAIL

POSTAGE AND FEES PAID
NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION

U.S.MAIL

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include conference proceedings, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546