NASA Contractor Report 178387

The Computational Structural Mechanics Testbed Architecture: Volume IV - The Global-Database Manager GAL-DBM

Mary A. Wright, Marc E. Regelbrugge, and Carlos A. Felippa

Lockheed Missiles and Space Company, Inc.
Palo Alto, California

Contract NAS1-18444

January 1989

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Preface

This Manual describes the late-1986 version of a data manager designed to operate on global-access data libraries (GALs). Data libraries are components of a *global database*. A global database archives data shared by independently executable applications programs, and survives excution of such programs. The structure of data libraries is hierarchical: libraries contain *datasets*, which in turn are collections of *records*. Records contain the actual data shared by the applications.

The particular database manager described in this document is called GAL-DBM. The first version of this manager, known as EZ-GAL, was written in 1980 to support the initial implementation phase of NICE (Network of Interactive Computational Elements). NICE is an integrated software system for computational mechanics under development at the Mechanics and Materials Engineering Laboratory at Lockheed Palo Alto Research Laboratory. NICE consists of *architectural utilities* and of independently executable programs called *Processors*. The architectural utilities support global and local data management, program-execution control by a command language, and source code maintenance. Operational compatibility is enforced by *replicating* the global data manager and command language interpreter in each processor.

GAL-DBM differs from an earlier version named EZ-GAL in that it deals exclusively with nominal-record capabilities.

Contents

1

INTRODUCTION

§1.1 NICE DATA MANAGEMENT

NICE, an acronym for Network of Interactive Computational Elements, is a database-coupled, executive-less, integrated software system under development since 1980 at the Mechanics and Materials Engineering Laboratory at the Lockheed Palo Alto Research Laboratory. NICE consists of *architectural components* discussed in ref. 1, and of computational elements called *Processors* that perform the useful work.

The NICE Data Management System (NICE-DMS) is one of three architectural components, the other two being execution control and source-code maintenance. NICE-DMS implements advanced techniques for the administration of scientific databases. Components of NICE-DMS are shown in Figure 1.1 interacting with other NICE components. The functional interaction of these components is discussed in §1.2; the following is just an overview.

NICE-DMS is a *layered* data management system. The qualifier "layered" means that the system consists of separable modules, with low-level modules serving higher-level ones. The two more important modules are in boxes 1 and 2 of Figure 1.1, labeled "I/O Manager" and "GAL-DBM", respectively.

The basic level of NICE-DMS is the Input-Output Manager (IOM), which is implemented as a package of FORTRAN 77 subroutines (complemented by assembly language subroutines on some computers) called DMGASP. The IOM functions as a modular interface between direct-access storage facilities (disk, core), and higher data management levels. Inasmuch as the I/O Manager interacts directly with the operating system, it is machine-dependent; however, this dependence disappears at higher levels. The present version of the I/O Manager is described in detail in ref. 2.

The present document is primarily concerned with the *Global Database Manager* part of NICE-DMS. A global database (GDB) is an organized collection of operational data that resides on permanent storage devices. These data are used by independently executable but logically interrelated application programs. Within the confines of the NICE architecture, these programs are known as *Processors*.

The NICE global database is constituted by sets of *data libraries*. The Global Database Manager of NICE-DMS is a package of FORTRAN 77 subroutines collectively called GAL-DBM. The main purpose of this Manager is to interface NICE Processors with *global access libraries* (GALs). A GAL is a data library that resides on a direct-access device such as a disk file (or even main storage) and complies with the technical specifications discussed in Section 2. GAL-DBM implements operations pertaining to the creation, access and modification of these libraries. The reader interested in a general functional description of the global database concept is advised to read ref. 3.

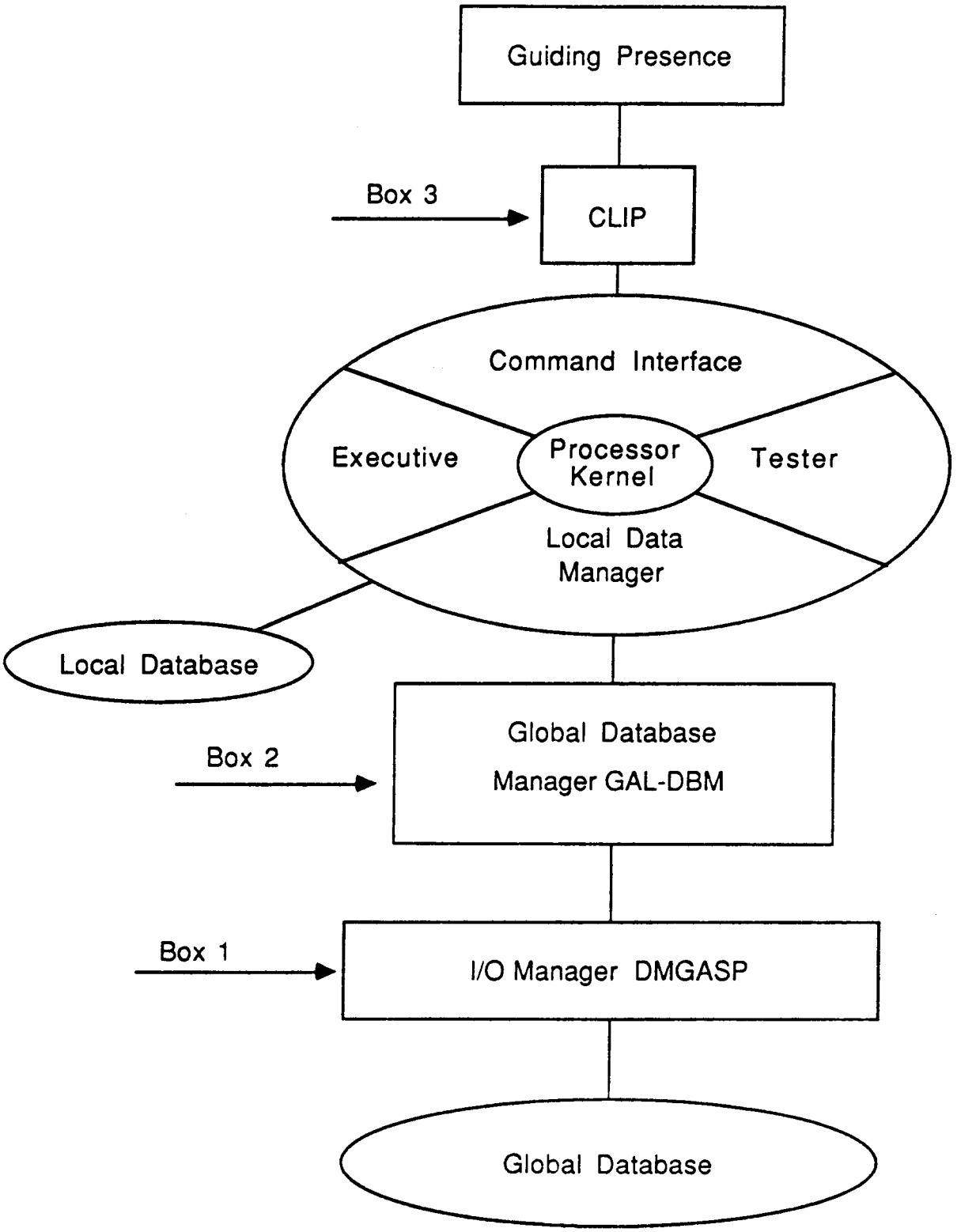

Figure 1.1. Configuration of a NICE Processor, showing data management and control components

§1.2 UTILIZATION OF GAL-DBM

§1.2.1. Making Processors Work Together

The essence of a network of data-coupled programs is that the output of one program becomes the input of another. This is an old idea popularized by the Unix system. NICE does belong to this very general network class, but it has its own structural and operational characteristics dictated by its intended application in computational mechanics.

How does GAL-DBM support the development of NICE Processors? This question is hopefully answered in Figure 1.1 which is a graphical representation of a typical Processor.

Processors have to be controlled in some way, and this is shown by a Guiding Presence. The Presence may be a human user interacting with a Processor, a command procedure written by a user, or a command procedure written by another Processor. But it is *never* a central executive program.

Interposed between the Processor and the Guiding Presence is the Command Language Interpreter Program CLIP (box 3 in figure 1.1), which is described in refs. 4-6. Interposed between the Processor and the global database is the Global Data Manager, which is shown separated into the GAL-DBM and DMGASP levels.

The Processor structure is shown in detail. A computational kernel is surrounded by a shell (similar to Unix). The shell contains four overhead components: Command Interface, Executive, Tester, and Local Data Manager. The construction of these shell components is left to the discretion of the Processor developer.

The Local Database embodies working data structures needed by the Processor itself to produce the results required by the Guiding Presence. Most of these data disappears when the Processor stops. The Local Data Manager is usually designed for maximal computational efficiency, and often custom-fitted to the Processor.

§1.3 CONTENTS OUTLINE

Sections 2, 3, 4 and 5 explain data libraries, datasets, indexed records and named records, respectively. These are the three data structure levels that GAL-DBM manages. Readers already familiar with data-library systems such as those used in the SPAR and DALPRO program networks may be able to absorb most of this material fairly quickly. Familiarity with ref. 2 would also be most helpful.

Section 6 presents library operations: open, close, flush and pack, which affect a data library as a whole.

Section 7 covers basic operations that affect one or more datasets and that do not depend on the record structure. For example: put dataset name in TOC (table of contents), find dataset, delete and enable datasets, list TOC.

Section 8 covers operations on indexed records that reside in positional datasets.

Section 9 covers operations on named records that reside in nominal datasets.

Section 10 covers supplementary and auxiliary operations that do not fit in the preceding four sections.

Section 11 covers information-retrieval functions.

Section 12 deals with copy operations.

Section 13 deals with text group operations.

Section 14 covers error handling in NICE-DMS in general and GAL-DBM in particular. It lists and explains error messages, and describes entry points that NICE programmers may use to access status information and to modify error-handling defaults.

2

Data Libraries

§2.1 OVERVIEW

What is a Data Library?

Webster's 7th Collegiate Dictionary defines a library as "a place in which books, manuscripts, musical scores, or other literary and artistic materials are kept for use but not for sale". The pessimist's view of a library has been quoted as "an organized collection of obsolete materials". However, the term *library*, or more precisely *data library*, is used here in a more specialized context. A library is an organized collection of data that possesses the following attributes:

1. It is intended (at least in principle) for use by independently executable programs.
2. It can be made available to a running program as a single logical entity.
3. It is oriented to the *archival* of data, rather than to the *processing* of data.
4. It provides at least one level of *named* access to the stored data and preferably more.
5. It is context-free, *i.e.*, no assumption as to the meaning of the stored data is made.

The qualifier *data* in *data library* is occasionally used to reinforce the distinction between data libraries and program libraries. The principal function of a program library is to catalogue source-code, object-code, or executable-code text. Program libraries are usually handled through operating-system utilities whereas data libraries are handled through application-oriented data management systems.

The chief function of data libraries is to serve as components of *global databases*. These are organized collections of information that are used as operational data for the applications system of a particular enterprise (*e.g.*, engineering analysis). Global database information is shared by functionally related but independently executable programs, and survives the execution of such programs. In a top-down description of scientific database structures, a data library may be defined as "a named partition of a database" (see ref. 5).

Nomenclature

Data libraries can be categorized according to the access characteristics of their residence medium into *direct-access and sequential-access libraries*. *Direct-access libraries* reside on direct-access storage devices such as disk or main-storage. *Sequential-access libraries* reside on serial-access devices such as magnetic tape.

All library organizations discussed in this document are of *direct-access* type. The use of direct-access online devices as a residence medium provides maximum operational flexibility, and is consistent with current computing industry practices. It should be noted, however, that permanent library files may (and should) be archived on tape during prolonged inactivity periods.

A data library contains *data objects* identified by names or numbers. Such data objects are the *members* or *elements* of a library. In the library organizations considered here, the

data structures associated with an individual member are either single records or sets of records. The term *dataset* (IBM-originated "computerese" for a collection of similar records related by spatial or temporal adjacency) will be used to describe such data structures.

Dataset description information such as identifiers, physical location and so on is usually abstracted into a special section of the data library known as an index, directory, catalog, data dictionary, or root segment. This section is usually implemented as a table (a matrix-like arrangement of information), which is then called the *Table Of Contents* or TOC.

The main reason for abstracting dataset information from the dataset records proper is efficiency in library-query operations. TOC segments can be brought to main storage through block-read operations and then searched at high speed. (The reader may think of the analogous situation in which a person tries to find a book in a public library: scanning an index is more efficient than walking through book stacks.)

The remainder of Section 2 is devoted to the description of the specific data library organizations processed by the GAL-DBM package, and to which the operations discussed in §§6-14 pertain.

§2.2 A CONCEPTUAL MODEL

To understand how GAL-DBM works, a conceptual model of data libraries is useful. Think of a data library as a bookshelf. The datasets are the books, and the dataset names are the book titles. For a data library containing three datasets: DENNIS, NELL, EDNA, this simple model can be diagrammed as:

```
DENNIS
    Dataset DENNIS
NELL
    Dataset NELL
EDNA
    Dataset EDNA
```

in which the distinction between the dataset name (the book title) and the dataset proper has been emphasized.

Now, what is a dataset proper? It is a collection of *records*. We shall define "record" precisely in §4.1; for the moment you may think of records as book chapters. But how are chapters identified?

In some books, chapters are merely identified by an index: 1, 2, 3, and so on, with perhaps a Preface, Introduction or Foreword which may be collectively called Chapter 0. On other books, you will find chapters identified by a *name* related to the subject matter. There is a close analogy for datasets. More precisely, datasets may be of two types: positional and nominal.

Positional Datasets. Records are identified by their sequence index: 0, 1, 2, ... Records must be physically contiguous; that is, record 2 must immediately follow record 1, and so on. Record 0 is called the *descriptor* record; its presence in a dataset is optional.

Nominal Datasets. Each record is identified by a name. The physical order in which records appear in the library is irrelevant. Most new software developments make use of nominal datasets exclusively.

To continue the example, suppose that that dataset EDNA contains five named records identified as LEON, NEDRA, ANITA, ROLF and NORA:

```
EDNA
    LEON
        Record LEON
    NEDRA
        Record NEDRA
    ANITA
        Record ANITA
    ROLF
        Record ROLF
    NORA
        Record NORA
```

Note that for dataset EDNA we have again emphasize the distinction between the record *name* and the actual record.

Can we go deeper? Book chapters are composed of paragraphs, which are composed of words. A word is the smallest component of our conceptual model, because if we break up a word into letters it loses meaning. The global-database counterparts to paragraphs and words are called *data aggregates* and *data items*, respectively. (In some cases, the aggregate level is missing or cannot be discerned.) Example: a full rectangular matrix is stored as a single dataset record; then the matrix columns are data aggregates while matrix entries are data items. Now if the matrix collapses to a row vector, aggregates collapse into items.

GAL-DBM is blind to aggregates and items. That is, the finest structure it can see is the record and it is blind to aggregates and items; these levels being of interest primarily to *local* data managers. All of this can be concisely expressed by saying that a GAL-DBM database has *record granularity*.

REMARK 2.1

The term *dataset*, in its original meaning, should strictly apply only to positional datasets, the records of which are related by physical adjacency. The proper name for nominal dataset is *group*, which is "a collection of records endowed with an owner or master record called the record directory, and a set of member records" (see ref. 5). But using different terminology for logically similar things tends to confuse users to no end. Purism has to give way to common sense, and so dataset is used. Besides, this frees the term "group" for use in describing a special amalgamation of named records in §5.

§2.3 LIBRARY DEVICES

Data libraries reside on *logical storage devices* under control of the I/O Manager DMGASP (cf. Figure 1.1). Most library devices are disk-resident permanent files, which naturally provide the permanency and direct-access attributes stressed in §2.1. But, as noted in §1.2, libraries can also reside on *scratch* devices to fulfill special needs.

The physical structure of library devices is outlined in §2.5, which should be of interest only to a minority of advanced users. In the following, only logical aspects are covered.

Library Identification

A data library has two identifiers: a name and a number. The nominal identifier is the *external device name* of the DMGASP device on which it resides. If the device is a permanent disk file, the device name either coincides with the file name, or it contains the file name in some fashion. For details the reader is referred to §2.6 and §3.1 of ref. 2.

The other identifier is the *Logical Device Index* (LDI) of DMGASP (see ref. 2). This is an integer in the range 1 through 30, which is linked to the external name when the library device is opened. The choice of LDI is arbitrary, but once selected, all subsequent references to the library are made through the LDI.

REMARK 2.2

GAL-DBM maintains a table of active libraries. The access pointer to this table is the *logical library index* (LLI). This index is only used internally, however, and is invisible to the user program. The decision of concealing the LLI was made to avoid confusing GAL-DBM programmers with the task of keeping track of several indices: one is enough. Furthermore, advanced programmers may sometimes want to access a library device at the I/O Manager level (*e.g.*, for a file dump); a common identifier is then beneficial.

Library Format

The present GAL-DBM can process three data library formats or *libforms*. These are identified as DAL, GAL80 and GAL82, respectively. The main characteristics of these libforms are summarized in Table 2.1.

The library form is established when a library file is created and cannot be changed afterwards. When an existing library is accessed, GAL-DBM can "sense" its form by scanning the library header.

DAL libforms are provided for compatibility with the DAL database. These libraries can reside only on disk files, are sector-addressable, and can accommodate only DAL-conforming datasets (a special positional dataset consisting of fixed-length records). Because of these restrictions, DAL libraries are not suitable for use by NICE processors, but do serve as a bridge to programs supported by DAL databases.

GAL libforms reside on word-addressable devices (disk files or core). The GAL80 form was provided by the original version of GAL-DBM; it can hold only positional datasets. The GAL82 form is provided by the present version and can hold nominal datasets.

Table 2.1. Libforms Processable by GAL-DBM

Format	*Addressing Unit*	*Residence Medium*	*Datasets Accepted*	*Comments*
DAL	Sector	Disk	DAL-conforming positional	Compatible with SPAR and DALPRO
GAL80	Word	Disk or core	Positional	Original GAL
GAL82	Word	Disk or core	Nominal	Latest GAL

§2.4 LIBRARY ACTIVITY OVERVIEW

Activation

GAL-DBM demands that any library to be made available to the user program be explicitly *opened.* The open operation, performed by GMOPEN (§6.4), connects a Logical Device Index to the external device name, and prepares the library for processing. An open library is said to be *active.*

If the library did not exist before the open operation, the open operation effectively combines creation and opening. A created library may be declared to be permanent if it is to survive closing (the most common case), or scratch if it is to disappear upon closing.

Although the Logical Device Index (LDI) ranges from 1 through 30, this does not mean that up to 30 libraries may be open at the same time. The present GAL-DBM allows up to *eight* libraries to be simultaneously active.

Write Protection

The access characteristics of the library are those of its resident device. The most common access restriction consists of making an existing library read-only. Some old-fashioned operating systems allow catalogued disk files to be declared read-only, so the file is unconditionally protected against all users. More modern operating systems, such as VAX/VMS, allow rings of selective protection.

But even if your host operating system does not provide write protection, read-only status can be specified at open time as explained in §3.1 of ref. 2. If so, the I/O Manager will disallow writes against the device.

Run-Abort Protection

The flush-library operation, performed by GMFLUB (§6.3), is used to safeguard new or modified libraries against abnormal run termination. This operation forces all core-resident buffers whose alterflag is on to be written to the library device.

Packing

Libraries used by heavily interactive programs such as pre- and post-processors tend to accumulate inaccessible datasets in the form of deleted datasets (§3.4). Library storage may be reclaimed by a *pack* operation, which is performed by invoking GMPACK (§6.5).

Deactivation

Once the user program is through (permanently or temporarily) with a library, it should release it with a *close* operation, which is performed by GMCLOS (§6.2). A close operation automatically flushes all main-storage buffers assigned to the library.

§2.5 PHYSICAL ORGANIZATION

A data library is usually stored as a permanent file on disk. Such files survive processor execution and thus provide the operational continuity expected from global databases.

The qualifier "usually" means that there are exceptions to the library-storage rule. A library of any type can reside on a scratch disk file, which disappears when it is closed. And libraries of GAL form can even reside on "core devices" provided by DMGASP in blank-common storage. Regardless of storage medium, a library is a *hierarchical indexed* organization. It contains at least two *overhead* data structures: a Library Header and the Table of Contents. ("Overhead" means that it is used only for internal administration.) In GAL82 libraries holding nominal datasets, a third overhead data structure appears: the Record Access Table (RAT), of which there is one per nominal dataset.

Library Header

The Library Header occupies the first 128 (64) words of the GAL (DAL) device. A short segment of the Header maintains state information such as the length of the library device, library format codes, and so on. The remainder contains the addresses of the TOC segments.

When an existing library is open, its header segment is read into a core-resident header buffer area, where it resides until the library is closed. If a header item changes as a result of an operation (*e.g.*, a TOC segment is created) the buffer is modified and an "alter flag" is set. Periodically the modified buffer is written ("flushed") to the actual library, and the alter flag cleared.

Table Of Contents

The Table Of Contents (TOC) maintains information about datasets. For example: dataset name, date and time of creation, dataset start location, number of records, and so on.

The TOC is not stored contiguously, but is subdivided into *segments*. Each segment is 384 words long and stores information about 16 (32) datasets in GAL (DAL) forms.

To process TOC segments, GAL-DBM maintains a TOC Buffer Pool in main storage with capacity for 2 to 4 segments (this number is declared as a FORTRAN PARAMETER at compile time). This (RAT) Buffer Pool is shared by all active libraries. Pool frames are allocated on demand using a LRU (Least Recently Used) paging policy. Segments altered as a result of database operations are marked as modified but not written back ("flushed") to the library until it becomes necessary to do so.

Record Access Tables

Each nominal dataset on a GAL82 library has a Record Access Table (RAT). The RAT configuration mimics that of the Library-Header/Table-Of-Contents pair. More specifically, the RAT is subdivided into 128-word segments; the segment addresses are stored in a RAT header that appears in front of the first segment. Each segment has information about 16 record entries (one ordinary record, or a record group).

To process RAT contents, GAL-DBM maintains a RAT Buffer Pool in main storage with capacity for 4 to 8 RAT segments. The Buffer Pool resources are shared by all active GAL82 libraries and all nominal datasets present in such libraries. As in the case of the RAT Buffer Pool, RAT segment frames are allocated on demand, using a LRU policy. Modified segments are alter-flag tagged, and not written back to the library device until necessary.

Dataset Storage

Indexed-record datasets occupy a *contiguous* area of the library device. So given the start of the dataset allocation, which is kept in the TOC, the location of a member record can be rapidly calculated if the record offset with respect to the dataset start is known. For record sets satisfying certain length constraints discussed in §4.2, the offset can be determined solely from TOC-stored information.

For nominal datasets, one more level is required. The dataset location field in the TOC points to the RAT Header, which holds locations of RAT segments. The location of a member record can be calculated from the base address and the item type and length information maintained in the RAT. This two-level addressing scheme allows member records to be physically anywhere in the library.

3

Datasets

§3.1 INDIVIDUAL DATASET IDENTIFICATION

Dataset Names

Datasets are identified by character strings known as *dataset names.* In its most general form, a dataset name consists of five components:

Mainkey.Extension.cycle1.cycle2.cycle3

where any component but the first may be omitted. Components are separated by periods. Examples:

```
MATRIX
A.B.1.3.2
S4-C1.ASM-STIF(G).3.65.1307
HELP.ADD
$TEMP$.BUFFER(+8)...4
RESPONSE..45
```

The total length of the name string, *including* connecting periods, must not exceed 40 characters.

REMARK 3.1

Two character key (alphanumeric) components and three integer cycle components are provided for maximum generality and flexibility in the naming of datasets.

Keys

The first two dataset-name components: *mainkey* and *key extension*, may contain any of the following characters:

letters A-Z, a-z
digits 0-9
dollar signs
plus and minus signs
left and right parentheses
underscore

Each key component, *excluding* connecting periods, is restricted to 16 characters. The key extension may be omitted, as in the first and last examples above. An omitted extension is conventionally the same as a *blank* character string.

REMARK 3.2

Key extensions that consist only of digits are permitted. For example:

```
MODE.139
```

Here 139 is a key extension, although it looks like a cycle number; it is *not* the same as MODE..139. To avoid confusing the poor user (or even yourself), please avoid all-digit keys.

REMARK 3.3

Upper and lower case characters are *not* equivalent. For NICE processors, the use of lower case characters in dataset names should be avoided, since the command interpreter CLIP normally converts automatically all character strings to upper case. Consequently, using lower case would impair the usefulness of command specifications that contain dataset names.

Cycle Numbers

The last three name components are called *cycle numbers.* As this term implies, a cycle is a sequence of digits that represents an *unsigned integer* in the range 0 through 99999. Any omitted cycle is assumed zero. Example:

```
ELEM.STIF.457
```

The first cycle is 457; the second and third cycles are zero. Similarly, `AA.BB` is the same as `AA.BB.0.0.0`.

Cycle numbers are primarily used for identifying datasets that differ only slightly, when such difference can be naturally associated with one index, an index pair, or an index triplet.

§3.2 MULTIPLE DATASET IDENTIFICATION

Certain library operations such as copy, delete, globally-match, list TOC, and print records, may affect more than one dataset. For such operations, it is often useful to be able to "bundle" together dataset names that are to participate in the operation. It is also occasionally useful to refer to things such as "the highest defined cycle number".

To facilitate these requirements, GAL-DBM provides *masking*, *cycle range*, and *relative cycle* capabilities. These are summarized in Table 3.1, and explained in the following subsections.

Name Component Masking

The simplest multiple-name-matching feature is *component masking*, in which a full name component is skipped when comparing against stored dataset names. This can be done with a "wild card" specification, in which a name component is replaced by an *asterisk*. Example:

```
DATA.*.*.2
```

This matches any identifier whose mainkey is DATA and last two cycles are 2 and 0. For example:

```
DATA.EPOXY.33.2
DATA..120.2
DATA...2
```

If the name ends with an asterisk, *i.e.*, the next character is a blank, everything that follows is masked. This is a special case of the trailing-asterisk rules stated below.

Partial Key Masking

More general masking specifications are possible on the mainkey and key-extension components. These work much like file name masking in the VAX/VMS operating system. The most common specification involves the use of a *trailing asterisk* to match name components with the same "root". For example:

```
F*.RUN.34
```

This name matches all datasets whose mainkey begins with F, followed by key extension RUN, and cycles 34, 0, and 0.

A *name-trailing* asterisk (*i.e.*, the next character is a blank) specifies matching against any combination of zero or more trailing characters. For example,

```
F*
```

matches all datasets whose mainkey begins with F, and

```
*.GA*
```

matches any dataset whose key extension begins with GA. Please note that this is not the same as

```
*.GA*.
```

which specifies zero cycle numbers. Thus, FUN.GAMES.3.4 is matched by the first form but not by the second.

In view of the trailing-asterisk rule, the five specifications

```
*.*.*.*.*
*.*.*.*
*.*.*
*.*
*
```

are effectively identical, and match *all* dataset names.

Leading asterisks are occasionally useful. For example,

```
RUN.*RE.*
```

matches all datasets whose mainkey is RUN and whose key extension ends with RE; *e.g.*, RUN.PROCEDURE or RUN.RE.67.8. The specification

```
*.*X*.*
```

matches all datasets whose key extension contains the letter X in any position.

REMARK 3.4

Asterisks which are neither trailing nor leading a key component (*i.e.*, an embedded asterisk) are not allowed. For example, NEW.ADV*LAM.6 is illegal.

Positional Masking

Individual characters at specific positions can be masked with the percent sign %, exactly as in VAX/VMS. For example:

```
DYN.RESP%%.*
```

matches any dataset with mainkey DYN and six-character key extensions beginning with RESP, *e.g.*, DYN.RESPON.5 or DYN.RESP02. Note the difference with

```
DYN.RESP*.*
```

which matches zero or more characters after RESP. For example,

```
DYN.RESPONSE
```

is matched by the second form, but not by the first one.

Cycle Ranges

Partial masking does not carry over to name cycles. For example

```
RUN.SET.67*
```

is meaningless, and will cause an error message to be printed. To match cycles that lie in a certain interval, the *cycle range* feature may be used. Example:

```
RUN.SET.4:67
```

matches datasets of the form

```
RUN.SET.cycle1
```

in which *cycle1* is 4 through 67 (inclusive). Cycle-range specifications may appear in more than one cycle, as in

```
RUN.SET.4:67.93.0:8
```

REMARK 3.5

The single asterisk specification in a cycle component is equivalent to the "match all" cycle range specification 0:99999.

Relative Cycle Specifications

In contexts where heavy use is made of cycle numbers, it is often useful to have compact ways of referencing things like "highest cycle number" or "next cycle number". GAL-DBM provides three letter symbols: L, H and N, that can appear as *first character* in *one* cycle field to form a *relative cycle* specification. The meaning is:

L	Lowest cycle number
H	Highest cycle number
N	Next cycle number (same as H+1)

These symbols may be optionally followed by a *signed* integer, as in

L+2	Lowest cycle plus 2
H-12	Highest cycle minus 12

To determine the numeric value of L, H or N, GAL-DBM does a TOC scan with the affected cycle field replaced by a mask. The appropriate value is substituted in an internal name string. If no match is made, L and H are conventionally set to zero whereas N is set to one.

Relative cycles may be combined with cycle-range specifications. The following are legal:

```
AA.BB.6.L:H
AA.BB.*.L+3:H-2
```

Relative specifications may not appear on more than one cycle component, however. Accordingly, the name

```
AA.BB.H.L+3
```

is illegal and will cause an error condition.

A common application of relative cycles is illustrated by the following example. A NICE processor creates results in a step-wise manner, and stores them in datasets named

```
RESULT.VEC.1
RESULT.VEC.2
RESULT.VEC.3
   ...
  etc.
```

so that the first cycle identifies the step number. After completing the 32nd step, say, the program creates the next receiving dataset using the name

```
RESULT.VEC.N
```

On receiving a name of this form, GAL-DBM scans the TOC for all datasets named RESULT.VEC.*cycle*1.0.0, finds out that the highest *cycle*1 is 31, adds one to produce 32, and so the dataset name winds up being RESULT.VEC.32.

Finally, suppose that the program is instructed to print the last three result datasets in the data library. The appropriate identifier is

```
RESULT.VEC.H-2:H
```

Table 3.1. Special Dataset Name Constructs

Name component	*Form*	*Effect*
Key	*	Matches zero or more arbitrary characters
Key	%	Matches one arbitrary character
Cycle	*	Matches any cycle number (same as 0:99999)
Cycle	$n_1:n_2$	Matches cycle numbers in range n_1 to n_2 inclusive
Cycle	H+n or H-n	H is to be replaced by highest cycle number (zero if none found)
Cycle	L+n or L-n	L is to be replaced by lowest cycle number (zero if none found)
Cycle	N+n or N-n	N is to be replaced by next cycle number (same as H+1)

§3.3 IDENTIFICATION BY SEQUENCE NUMBER

Each dataset, whether enabled or deleted (§3.4), has a unique *sequence number*, which is the ordinal of its occurrence in the data library. Once a dataset is entered in a library, its sequence number cannot change unless the library is packed.

Most GAL-DBM operations identify a dataset by its sequence number rather than by name. This has the following advantages:

1. No search of the library Table of Contents is required. Such a search can be expensive in terms of CPU time and I/O accesses when a library contains over 100 datasets.

2. Datasets with identical names may be told apart, which is useful for "undelete" or "copy deleted" operations.

3. Sequence number range specifications may be given. These are handy for requests such as "list the last 10 dataset names".

4. In interactive command input, small integers can be typed faster than long names.

Connections between names and sequence numbers are usually established through "find name" search operations.

§3.4 DATASET STATES

During its lifetime, a stored dataset may be in one of three states:

Enabled unlocked. A dataset can be accessed and operated upon with no restrictions of any kind.

Enabled locked. A dataset flagged with access or use constraints (more details below).

Deleted. A deleted dataset is physically present in the data library, but is "transparent" as regards "find" operations. A deleted dataset cannot be directly operated upon. It can be rendered accessible only through an explicit "undelete" (enable) operation.

Locking

Access restrictions on enabled datasets may be specified by the user program through a *lock code*, which is maintained in the TOC. The lock code of an unlocked dataset has a zero value, which is the default state, while nonzero values are used to define various protection levels. Please refer to Table 3.2 for a detailed explanation.

Two situations in which locking is useful should be mentioned.

Questionable data.. A dataset may be produced under *abnormal conditions*; for example an iterative solution process may fail to converge, but the last iterate is saved anyway. A lock code of -1 (cf. Table 3.2) comes handy in this situation.

Be prepared. A NICE processor may want to safeguard valuable information stored in certain datasets by protecting them against write-in-place, extend or delete operations performed by itself or by other NICE processors.

REMARK 3.6

The lock code feature has not been fully implemented.

Deletion

The most common source of deleted datasets is the insertion of a dataset name that duplicates an existing name. The general rule is: *names of enabled datasets must be unique*. For example, suppose that dataset DENNIS is entered thrice in a library. The logical configuration would then be

```
*DENNIS
    Dataset DENNIS
*DENNIS
    Dataset DENNIS
DENNIS
    Dataset DENNIS
```

in which asterisks flag deleted datasets. If now the user program "undeletes" (enables) the first dataset, the third one is automatically marked as deleted because of the uniqueness rule.

Table 3.2. **Dataset Lock Codes**

Lock code	*Protection level*	*Explanation*
0	None	Dataset may be extended, rewritten or deleted
1	Read-only	Dataset may be deleted, but not extended or rewritten
2	Undeletable	Dataset cannot be extended, rewritten or deleted
3	Untouchable	Same as 2 and in addition a lock code can be subsequently lowered (to 0, 1 or 2) only by the *generating processor*. More precisely, the processor name declared through GMSIGN must match that stored in the TOC for a lower protection level be installable
-1	Warning message	Same as 1 and in addition a warning message is printed each time dataset records are read. Useful for flagging erroneous data stored in the global database.

§3.5 DATASET ACTIVITY

Dataset Creation

A dataset is created in a library by a *put name* operation, which installs the dataset entry in the TOC. If the dataset is of nominal type, the operation also creates the first segment of its Record Access Table (RAT) and its Header.

For a *positional* dataset, the put-name operation is often followed by a *reserve-space* request, which sets aside enough space to hold all of the dataset records. For *nominal* datasets, this is never necessary.

Record Transmission

This topic is covered in §4 and §5.

Dataset Deletion

To explicitly get rid of a dataset, a two-step process is necessary. First, the dataset must be marked as deleted. Then the library is packed. The second step physically removes the space used by the dataset. In practice, the library-packing step is done only occasionally, as it is quite expensive.

4

Indexed Records

Data libraries are collections of datasets, and datasets are collections of records. Records hold the applications data that feeds Processors. As noted in §2.2, GAL-DBM has essentially record granularity; so this Manual does not contain sections explaining data aggregates and items.

This Section deals with aspects of global data management related to the record structure of datasets. §4.1 gives an overview of what records are and how record identification characterize datasets and libraries into positional and nominal. Then §§4.2-4.3 give further details on indexed records in positional datasets. Named records are discussed in §5.

§4.1 OVERVIEW

Definition

The technical definition of *dataset record* is

> A string of data objects that occupies adjacent locations in the library device, belongs to one and only one dataset, and is identified by number or by name.

Grasping the concept of "adjacent locations" requires familiarity with the physical structure of direct-access devices (as explained, for example, in §2 of ref. 2). The GAL-DBM programmer is generally more interested in the functional characterization:

> A dataset record is a string of data that belongs to one and only one dataset, is identified by number or name, and can be read into main storage as a gapless, undivided whole.

This is a more useful characterization. Basically, the GAL-DBM programmer wants to say "write record", and the record goes away, and then "read record" and the record comes back unspoiled (PL/1 and Pascal programmers may think of "put" and "get", but the effect is the same).

REMARK 4.1

The functional definition does not imply that a dataset record is stored in a single write operation, although that is often the case. A long indexed record may be built "by chunks" provided that no gaps can occur between the chunks. In more technical terms, the chunks are called record blocks, and a dataset record built in this fashion is called a *block-spanning record*, or spanning record for short.

REMARK 4.2

Dataset records do not have any end-of-record marks. This is in line with the philosophy of viewing direct-access devices as simply a linear array of storage locations.

REMARK 4.3

Users of the I/O Manager (IOM) DMGASP may wonder about the relation between IOM records and dataset records. The answer is that each dataset record is built up of one or more IOM records (which is not at all surprising, since GAL-DBM channels all physical I/O through DMGASP). But the converse is not true. Dataset records are closer to the *logical* level than IOM records.

§4.1.2. Classification

It was noted in §2.2 that there are two ways of identifying dataset records:

1. *Indexed records* are physically contiguous, and are identified by sequence number.

2. *Named records* need not be physically contiguous, and are identified by name.

Indexed and named records may not be mixed within the same dataset or even the same library. Consequently, datasets and libraries are naturally categorized into two types: positional and nominal. The type is selected when the library is created, and cannot be changed afterwards.

For a positional dataset, the physical order of records usually corresponds to the chronological order of acceptance of those records by the data manager. Accordingly, the record identification is intimately related to the definition order. On the other hand, for a nominal dataset the order in which records are defined is irrelevant.

The operational characteristics of positional datasets are discussed in the following subsections. Positional datasets should be used only by programs that are to maintain some form of compatibility with DAL databases. Otherwise, the nominal-dataset organization should be preferred, as it offers more flexibility and growth potential.

§4.2 POSITIONAL DATASETS

The adjective that best describes a positional dataset is *tape-like*. Records are defined by their position within the dataset, and are physically contiguous. Once the next dataset is installed, a positional dataset cannot be further expanded unless sufficient space has been reserved in advance.

The most important types of positional datasets are those called DAL-conforming and GAL-conforming, respectively, which are described below.

DAL-Conforming Datasets

A DAL-conforming dataset is an indexed-record stream that meets the following conditions.

1. All records, with the possible exception of the last one, have the same length. The length of the last record may be equal to or smaller than the others.

2. There is no descriptor record.

3. Records contain items of identical type. (See Remark 4.4).

To illustrate these rules, consider a dataset that consists of six floating-point records whose lengths are 60, 60, 60, 60, 60 and 42 words. This is a DAL-conforming dataset. If the size of the last records were 61 words or more, it would not qualify. If the first record contained integers, it would not qualify either.

The internal contents of these datasets is fully described by the type code stored in the library Table of Contents (shown under "T" in TOC listings). The type code is an integer that may assume the values listed in Table 4.1.

DAL-conforming datasets are accepted by programs that make use of DAL files for local or global data management. At the Lockheed Palo Alto Research Laboratory, these programs include DALPRO, NEPSAP, and SPAR.

DAL compatibility means that these datasets can be copied to DAL files, processed by utility programs such as DALPRO, and copied back to GAL files without critical loss of information. Content homogeneity implies that they are comparatively easy to move from one computer to another, inasmuch as item-by-item conversion is simplified.

REMARK 4.4

The homogenous-data-type rule is largely enforced within the DALPRO program, which deals primarily with rectangular matrix structures. However, many datasets used by the SPAR network do violate the rule.

Text Datasets

These are special DAL-conforming datasets that store *card image* data. They are identified by DAL type codes 5 and 6 (Table 4.1). Both types are extensively used by the command interpreter CLIP (see refs. 1, 4-6) to store command procedures, and CLAMP scripts.

Table 4.1. **Datatype Codes for DAL-Conforming Datasets**

Code	*Data type of record items*
0	integer
±1	single-precision real
±2	double-precision real
±3	single-precision complex
±4	alphanumeric (SPAR format)
±5	variable-length-line card images (DALPRO format)
±6	fixed-length-line card images (DALPRO format)
7-9	reserved for DALPRO use
10-higher	available to DAL programmers

Data type 5 stores variable-length card images. All rightmost blanks are stripped and a character counter is stored in front of each image. These images are *blocked* into records of equal size (usually 256 or 512 bytes). This storage arrangement is space-efficient but not suitable for random access of individual images. It is used for CLIP command procedures, CLIP ADD elements, and DALPRO runstreams.

Data type 6 stores fixed-length card images (80 characters), one image per record. This arrangement is space wasteful, but simplifies random access to individual card images. It is used for the "compiled form" of CLIP command procedures, in which the random access feature is necessary to take care of branching to labeled commands.

GAL-Conforming Positional Datasets

A GAL-conforming positional dataset is a stream of indexed records that complies with the following rules.

1. The first, second, third and last record lengths may be different from the others. The fourth through the next-to-last record must have the same length. Obviously, any dataset with five records or less satisfies this constraint.

2. A character descriptor record (record 0) of arbitrary length is permitted.

3. Each record must be either numeric or character. Mixing of character and numeric data within the same record is forbidden.

4. Mixing of different numeric data types within the same record (for example, integers and floating-point numbers) is permitted but discouraged.

As an example, consider a 162-record dataset that consists of

Descriptor:	120 characters
Record 1:	20 integers
Record 2:	240 reals
Record 3:	144 integers
Records 4-161:	642 double-precision floats
Record 162:	288 reals

This is a GAL-conforming dataset.

The record-length rule is the most important one. It allows the location and size of each record to be calculated from information stored in the Table of Contents. This information includes: location, total dataset size, length of descriptor, first, second, third, and last record. The common length of the fourth-through-next-to-last records, if any, is obtained by subtracting the sum of other record lengths from the total dataset size (descriptor excluded) and dividing by the number of records minus four.

The length of numeric records, which form the overwhelming majority, is always expressed in *machine words*. The length of character records is expressed in characters, but internally they are blank-filled to the next machine word boundary.

Because of their greater generality, GAL-conforming positional datasets rely on the descriptor record and leading control records for self-description. This descriptive information, however, has to be interpreted *outside* GAL-DBM, which leads to problems (see *Context sensitivity* below).

GAL-conforming datasets provide more operational flexibility than DAL-conforming ones. The global manager is still able to access any dataset record with minimal overhead, simply from TOC-stored information.

REMARK 4.5

The NIFTY-formatting conventions are based on the notion of *templates* (see reference 7). A template is the storage layout of a GAL-conforming positional dataset proven useful for various applications. Template datasets do have reserved leading records called *control packets*, which store data that describe the logical structure of the records that follow.

Assessment

Advantages of positional datasets center on processing efficiency. In particular:

Low storage overhead. For a positional dataset, the only storage overhead is its TOC entry, which amounts to 24 words for GAL forms, and 12 words for DAL forms, respectively. This overhead is independent of the number of stored records.

Low access overhead. If the dataset sequence number is known, as is the case in most read/write sequences, any record can be accessed with at most two *logical* device accesses: one to get the pertinent TOC segment into main storage, and another to get the actual record. If the TOC segment happens to be in the TOC Buffer Pool, only one logical access is required. (The number of *physical* accesses at the I/O Manager level may be less or more than this, depending on whether the library device is paged or not, record alignments, etc.)

Multirecord access. Several adjacent records can be read in one call, thereby further reducing access overhead. (This is restricted to word-addressable GAL libforms however, because in a DAL file, gaps can appear between records.)

DAL compatibility. DAL-conforming datasets provide a useful bridge to the SPAR and DALPRO programs, and their rich set of utilities (*e.g.*, graphics packages).

Disadvantages of positional datasets center on their proximity to the physical storage level. More specifically:

Ordering Sensitivity. The tape-like character makes positional datasets difficult to set up should records be generated in non-sequential order. For example, suppose that record

2 is the first one generated by the processor. Then appropriate space for the descriptor and record 1 has to be reserved by writing dummy records.

Extension Difficulties. Appending records is easy if the dataset is the last one in the library. If not, sufficient space has to be reserved in advance to allow worst-case expansion. For volatile datasets (those whose final size cannot be closely predicted in advance), this can be a serious problem.

Lack of Internal Self-Description. The generality of GAL-conforming datasets coupled with limited TOC space means that knowledge about data type (such as precision information) has to be kept *outside* the data manager. This forces processor developers to write many specialized utilities. It also makes movement of datasets between heterogenous computers a cumbersome operation.

Weak typing. Length information about numeric records is expressed in machine words. Processing multiword items such as double-precision and complex data leads to awkward interface calls.

Context Sensitivity. The meaning of every piece of data is strictly determined by position. For homogenous structures such as rectangular matrices and vectors, this feature is irrelevant. But for heterogenous datasets, positional sensitivity can be an invitation to disaster. Just displace an item one slot, and watch a program network collapse.

§4.3 INDEXED RECORD OPERATIONS

This section summarizes the basic operations on positional datasets. The GAL-DBM entry points named here are discussed in §§7-8.

Append

A positional dataset is extended by one record. This takes three steps:

(a) If the dataset sequence number is unknown, find the dataset through LMFIND (§7.5).

(b) Position to end of the dataset through GMFEND or LMFEND (§8.2).

(c) Write record through GMTRAN or GMTRAC (§8.6).

Potential problems: lack of space to receive record. This can be checked before step (c) by testing the return of LMFEND against the record size. If there is not enough space, copy the dataset to the end of the library, then repeat steps (a) through (c).

Find

Step (a) is as above. Then position to desired record through GMFIRE or LMFIRE (§8.3). The latter entry returns the record size, if it exists.

Read

Perform a find-record operation through LMFIND. If the record exists, move it to main storage through GMTRAN or GMTRAC (§8.6).

Rewrite

Perform a find-record operation, as in the read-record case. After checking size if necessary, follow with write-in-place through GMTRAN or GMTRAC (§8.6) and the contents are replaced.

5
Named Records

§5.1 NOMINAL DATASETS

Record Identification

A named record is identified by a name rather than by position. The record name consists of two components: a symbolic key and a cycle, separated by a period. For example:

```
EDNA.345
```

Here EDNA is the record key and 345 is the record cycle.

The key is a string of up to 12 characters. Legal characters are exactly those accepted for dataset keys (§3.1). The cycle is a nonnegative integer in the range 0 through 99999.

If the cycle is zero, it may be omitted. Thus

```
LINDA.0
LINDA
```

are the same thing.

All items in a named record should have the same datatype. Across the GAL-DBM/user-program interface, the datatype is defined by the *external* one-character codes listed in Table 5.1. Within the database, the *internal* integer code specified by the NIFTY[7] standard is stored. No provision is made for the type LOGICAL, which is a FORTRAN anachronism. The "unknown" type is returned to the user program in multiple-record retrieval operations if accessed records have different types.

Lengths of named records are expressed in *logical* units, *i.e.*, by item count. For example, the length of a named record consisting of 200 double-precision items (external type D) is 200, rather than 400 words as it would be for an indexed record. Table 5.1 correlates external type codes to physical storage units.

Advantages and Disadvantages

Nominal datasets offer the following operational advantages over positional ones:

1. Any combination of record lengths can be used, and will be correctly handled by the data manager. The constraints noted in §4.2 for DAL-conforming and GAL-conforming datasets do not apply.

2. Records may be defined in any order, as their relative position has no significance.

3. Nominal datasets may be extended with new records at any time (unless the dataset is locked or deleted) because the records need not be physically contiguous. A new record is simply stored at the end of the library, and a link to it placed in the Record Access Table.

Table 5.1. **Datatype Codes and Storage Units for Named Records**

Datatype of record items	*External code*	NIFTY[7] *code*	*Storage Units per record item*
Integer	I	0	1 word
Single-precision floating-point	S	1	1 word
Double-precision floating-point	D	2	2 words
Single-precision complex	C	3	2 words
Character	A	5	1 character
Mixed	M	-	1 word
Unknown	U	-	1 word

4. The record datatype is maintained by the manager. This simplifies context-directed display operations, automatic type conversion, and moving data between different computers.

5. The use of logical lengths leads to more readable coding.

6. The record-transmission interface is cleaner, and yet far more powerful. For example, many names can be made to point to the same record, which elegantly takes care of things like repeated element stiffnesses in finite element analysis.

These are certainly attractive features, but they are not without cost. The superior flexibility of named records is paid for in terms of additional disk storage (8 words per record in the worst case), and additional disk accesses incurred in searching the Record Access Table. These overhead costs can be considerably reduced, however, through the use of record groups and tables, as explained in the following subsections.

Record Groups

Many scientific data structures involve regular record occurrences. More precisely, records of equal length and type. In such cases, the records may be *grouped* under a common key to form a *Record Group* or *Group* for short.

A record group is identified by a symbolic key and a *cycle range*. Example:

```
ELLEN.66:145
```

Here ELLEN is the record-group key whereas 66:145 specifies the cycle range; 66 is called the *low cycle* while 145 is the *high cycle*. The example Group contains $145 - 66 + 1 = 80$ records of identical size and type. If each record contains, say, 12 items, the total Group length would be 960 items.

If the low and high cycles coincide, the record group reduces to an ordinary named record. Thus

```
LORA.4:4
LORA.4
```

represent the same thing.

Individual records within a Group may be accessed as one ordinary record. Thus, for ELLEN.66:145,

ELLEN.66	identifies the 1st record
ELLEN.88	identifies the 23rd record

Consecutive records may be accessed by specifying a cycle subrange; for example,

ELLEN.76:85

specifies records 11 through 20 (inclusive) of ELLEN.66:145.

Why grouping? It offers potentially huge savings in storage and access overhead. A Group of n records occupies only *one* entry in the Record Access Table, rather than n entries. A specific case is discussed later in this Section.

Tables: Basic Concepts

A *table* is a two-dimensional, matrix-like arrangement of data. The following simple example will be used to illustrate basic concepts:

NODE	X-COOR	Y-COOR	Z-COOR
n_1	x_1	y_1	z_1
n_2	x_2	y_2	z_2
n_3	x_3	y_3	z_3
n_4	x_4	y_4	z_4
n_5	x_5	y_5	z_5
n_6	x_6	y_6	z_6

The meaning of this table would be obvious to a finite-element or finite-difference developer, but is not relevant to the ensuing discussion. Much of the terminology that follows comes from the theory of *relational databases*, in which tables play a fundamental role.

Columns of a table are called *attributes*. Each column is identified by a unique *attribute key*. In the example table, the attribute keys are NODE, X-COOR, Y-COOR and Z-COOR. Matrix rows are the table *entries*.

The order in which columns appear is irrelevant. For example, the following table is equivalent to the previous one:

Y-COOR	NODE	X-COOR	Z-COOR
y_1	n_1	x_1	z_1
y_2	n_2	x_2	z_2
y_3	n_3	x_3	z_3
y_4	n_4	x_4	z_4
y_5	n_5	x_5	z_5
y_6	n_6	x_6	z_6

Incidentally, this shows that tables and matrices are not equivalent. Permuting columns of a matrix produces a different matrix.

REMARK 5.1

A *normalized table* is a table in which: (a) each table entry is atomic (a nondecomposable item), and (b) no two rows are identical. Normalized tables are extremely important in relational database

theory, where they are called *relations*. Rows of a normalized table are called *tuples* (pronounced to rhyme with couples) in relational database literature.

GAL-DBM Tables

The concept of Table in GAL-DBM is quite general and is not restricted to normalized tables (Remark 5.1). A Table (with capital T) is a *two-dimensional arrangement of named records* that meets the following requirements:

1. Column-aligned records have identical size and datatype.
2. Column-aligned records are identified by the attribute key as a common record key.
3. Rows are identified by the row index used as a record cycle.

For example, suppose that the x_i, y_i and z_i entries of the example table are merged into 3-item records, and the resulting attribute called XYZ-COOR:

NODE	XYZ-COOR
n_1	x_1, y_1, z_1
n_2	x_2, y_2, z_2
n_3	x_3, y_3, z_3
n_4	x_4, y_4, z_4
n_5	x_5, y_5, z_5
n_6	x_6, y_6, z_6

This table is a valid GAL-DBM Table, although no longer a normalized one. Now `XYZ-COOR.5` means the triple (x_5, y_5, z_5).

REMARK 5.2

If each record component reduces to one item, and row duplications are precluded, an GAL-DBM Table reduces to a Normalized Table. From this it follows that GAL-DBM is suitable as an *archival* system for ordinary relational data managers. But of course it is not restricted to such a provincial function.

Table Representation with Record Groups

The definition of GAL-DBM Table suggests immediately that each Table column be identified as a record group. Because of the row-identification rule, the cycle range of these Groups must be the same. This motivates the following (indirect) definition.

A set of record groups with *identical cycle range* and which resides in the *same nominal dataset* may be logically referenced as a Table whose attribute keys are the Group keys. Thus, to represent the example table of §4.4.4, define the four Groups:

```
NODE.1:6
X-COOR.1:6
Y-COOR.1:6
Z-COOR.1:6
```

A segment of a Table is specified by concatenating record keys with the ampersand operator, followed by the row range. Thus

```
NODE&X-COOR&Z-COOR.2:4
```

specifies the following segment of the example Table:

NODE	X-COOR	Z-COOR
n_2	x_2	z_2
n_3	x_3	z_3
n_4	x_4	z_4

The name

```
NODE&X-COOR&Y-COOR&Z-COOR.3
```

identifies the 3rd row. Finally,

```
NODE&X-COOR&Y-COOR&Z-COOR.1:6
```

identifies the complete Table. This naming convention is primarily used when an existing Table is accessed for GET and PRINT operations.

Transposed Tables

A table whose attributes are *stored* row-wise is called a *transposed table.* The transposed form of the example table above is

NODE	n_1	n_2	n_3	n_4	n_5	n_6
X-COOR	x_1	x_2	x_3	x_4	x_5	x_6
Y-COOR	y_1	y_2	y_3	y_4	y_5	y_6
Z-COOR	z_1	z_2	z_3	z_4	z_5	z_6

A column of a transposed table is a data aggregate sometimes called a *packet* in the literature. This term has the connotation that items of different type are "bundled" by physical proximity.

Storing a table in transposed form does not change its intrinsic meaning: it just looks different. The transposed arrangement is commonly used in FORTRAN codes to simplify the assembly of tables when the number of entries is highly volatile (these are sometimes called dynamic tables). This arrangement is a result of FORTRAN array-storage conventions: appending a column to a two-dimensional array is easier than appending rows (which require a change in the first array dimension).

To save a transposed table as an GAL-DBM Table, rows are made into Record Groups. Inasmuch as row items are not contiguous in main storage, an *increment* specification has

to be given to the record-group-writer routine. The same goes for reading Group items into non-adjacent locations.

Implementation Considerations

As noted in §2.5, the Global Data Manager keeps track of named records through a Record Access Table (RAT), which is in fact a transposed table. There is one RAT for each nominal dataset. The RAT structure strongly resembles that of the Table of Contents (TOC): it is partitioned into segments with a short segment-address header at the front of the first segment.

When a new nominal dataset is opened, the RAT header area and the first RAT segment are allocated and initialized. As records are defined, they are allocated RAT entries called *packets*. Once a RAT segment is filled, a new segment is allocated at the end of the library, initialized, and its address placed in the RAT header.

The Record Access Packet contains the following information: key, low and high cycle (these being coincident for an ordinary record), datatype, device address, logical length, physical offset, and row dimension. The offset item is only used in Groups and Tables to calculate the location of member records. The row dimension is only used for named records that are declared as rectangular matrices.

The mechanics of record accessing can be illustrated with a simple example. Ordinary record `EDNA.65` is requested, `EDNA` not being a Group key. To find a record, the Data Manager has, in principle, to scan the entire RAT. If any RAT segments for the owner dataset happen to be in the RAT Buffer Pool, those are searched first. If not found, RAT segments are brought to main storage in chain-like order, and searched.

Now suppose that the dataset under question contains 3200 ordinary records. If the record being searched for is in the RAT and access requests are randomly placed, 1600 RAT entries will have to be examined on the average, and 100 RAT segments searched. This process can add up to a lot of disk accesses. Also note that the 3200 RAT entries take up $3200 \times 8 = 25600$ words of storage (200 disk blocks on a VAX), which is not a negligible amount of storage overhead.

Next, suppose that `EDNA.65` is part of a record group `EDNA.1:3200`, which is the only thing in the dataset. The RAT will contain only *one* entry, and the record will be found on the first try, not 1600. The storage overhead will be only 128 words (one RAT segment) rather than 25600, a 200:1 improvement ratio.

This example clearly illustrates the dramatic efficiency advantages that can accrue from the use of record grouping.

Record Groups vs. Indexed Records

Since record groups are simply collections of equal-size records, a perceptive reader might ask: why not use indexed records and avoid the RAT overhead completely?

If a dataset is to contain simply one Group, the answer is that it does not make much difference, and if you are desperate for maximum efficiency, a positional organization might

as well be chosen. However, the occurrence of any of the following factors tilts the balance toward a nominal organization:

1. The dataset is to contain *several* Groups, possibly intermixed with ordinary records (*e.g.*, control records).

2. Full data self-description is important, for example if the dataset is to be moved to another computer.

3. The final extent of the dataset is not known in advance.

The last factor is of considerable importance in highly interactive programs such as graphic pre- and post-processors. Consider the following example.

An interactive geometry pre-processor defines sets of points identified by a user number (N) and three Cartesian coordinates (X, Y, and Z). The processor developer decides to put all of this information into one nominal dataset with four Groups:

N.1:*n*, X.1:*n*, Y.1:*n*, Z.1:*n*

in which *n* denotes the number of points defined by the user. The trouble is, *n* is not known in advance. The developer decides to split these Groups into groups of 100 records each. Thus at the start of the interactive session, the four Groups

N.1:100, X.1:100, Y.1:100, Z.1:100

are defined and initialized. When the user defines the 101-th point, four more groups are defined:

N.101:200, X.101:200, Y.101:200, Z.101:200

and so on. The fact that nominal dataset expansion is trivial, regardless of what other database transactions may have occurred in the interim, makes named records particularly attractive.

Note also that each record consists of only one item. If record grouping is not used, the overhead would be intolerable, *e.g.*, 6400 words for 200 sampling points! With record grouping, the overhead is only 64 words for 200 points, which is quite reasonable.

6

Library Operations

§6.1 GENERAL DESCRIPTION

Four GAL-DBM operations: open library, close library, flush library and pack library, apply to a data library as a whole (*i.e.*, are not concerned with individual datasets). Associated entry points are alphabetically ordered in this Section by the last four letters of the entry point name. A summary list is provided in Table 6.1.

Most programmers should be familiar with the first two operations. *Open* makes the library device (resident on a disk file or main storage) available for processing and performs various initializations. *Close* terminates library processing and releases associated storage resources.

Flush is a more specialized operation: it forces modified library-table, core-resident buffers to be written to the disk-resident library file. This operation is a precautionary measure against abnormal run terminations, because a *Close* operation on normal termination automatically flushes all buffers.

Finally, *pack* physically eliminates all deleted datasets from a library. This operation is the only GAL-DBM operation that may change dataset sequence numbers. Accordingly, it should be used with extreme caution if performed by a running program that assumes invariance of sequence numbers.

REMARK 6.1

Any user-program subroutine that references one of the following entry points should first identify itself by calling **GMUSER** as explained in §14.4. This information is used by the central error management routine of NICE-DMS for traceback prints.

Table 6.1 **Entry Points for Library File Operations**

Operation	*Entry Point*	*Arguments*	*See*
Close library	GMCLOS	LDI, 0, TRACE	§6.2
Flush library	GMFLUB	LDI, 0, TRACE	§6.3
Open library	GMOPEN	LDI, EDNAME, DDPARS, LBFORM, TRACE	§6.4
	LMOPEN	KEY, 0, EDNAME, LIMIT, TRACE	§6.4
Pack library	GMPACK	LDI, 0, TRACE	§6.5

§6.2 CLOSE LIBRARY: GMCLOS

This operation breaks the connection between a Logical Device Index (LDI) and the data library. The storage resources are released to the operating system, and cease to exist if the library device was of scratch type.

FORTRAN Reference

Calling Sequence

```
CALL GMCLOS (LDI, 0, TRACE)
```

Input Arguments

LDI	If > 0, Logical Device Index of library device to be closed. If this LDI is not active, no operation is performed. If zero, close *all* open libraries. LDI < 0 means *conditional* close. If the "NICE macroprocessor" flag has been set to on using GMACRO (§10.2), the close request is ignored, but the flush is performed. Otherwise device |LDI| is closed.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 6.2

A nonzero second argument is used in internal calls.

REMARK 6.3

If the library resides on a permanent disk file, GMCLOS flushes its core-resident buffers by calling GMFLUB (§6.3) before closing the file.

EXAMPLE 6.1

```
CALL GMCLOS (7, 0, 1600)
```

This statement closes the library connected to Logical Device Index 7.

EXAMPLE 6.2

```
CALL GMCLOS (0, 0, 1600)
```

This statement closes all open libraries.

Directive Reference

*CLOSE [*ldi*]

If *ldi* is specified, this directive closes the library attached to Logical Device Index LDI. If omitted, zero is assumed and all active libraries are closed. A negative *ldi* has the same interpretation as in the FORTRAN reference.

EXAMPLE 6.3

```
CALL CLPUT (' *CL ')
```

This message causes all active libraries to be closed.

Close Message

The DMGASP close service (see ref. 2, §3) writes an informative message on the bulk-print file. For an auxiliary-storage device, the format is typified by the example

```
<DM> CLOSE, LDI: 8, File:  RES.GAL
```

which is self-explanatory.

REMARK 6.4

As in the case of the OPEN message (§6.4), the CLOSE message is written out before the service is actually performed, and an error diagnostic may conceivably follow. However, close-file errors are comparatively rare.

REMARK 6.5

On Univac, the message format for a *Block I/O device* is different: it will show the @FREE image submitted to the Exec-1100 operating system. For FORTRAN I/O devices, the message has the standard form shown above.

REMARK 6.6

In the case of a conditional close, no message appears if the operation is skipped.

REMARK 6.7

Open and close messages may be altogether suppressed by calling GMSOCM (§10.9).

§6.3 FLUSH LIBRARY: GMFLUB

This operation forces a new or modified library to be *properly configured*, meaning that the Library Header, Table Of Contents, and Record Access Tables stored in the library file reflects the current library state. To achieve conformity, GMFLUB scans the TOC and RAT Page Buffer Pools, and writes altered pages to the library device. The library is *not* closed.

FORTRAN Reference

Calling sequence

```
CALL GMFLUB (LDI, 0, TRACE)
```

Input Arguments

LDI	If LDI > 0, Logical Device Index of library device to be flushed. If LDI = 0, all open libraries are flushed.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 6.8

A nonzero second argument is reserved for internal calls.

REMARK 6.9

No operation is performed if the library has not been written on since the last **GMFLUB** call, or the library-open operation, whichever occurred last.

REMARK 6.10

If the library resides on a scratch file, or on a core device, the flush operation is ignored.

EXAMPLE 6.4

```
CALL GMFLUB (8, 0, 1600)
```

This statement flushes the library connected to Logical Device Index 8.

EXAMPLE 6.5

```
CALL GMFLUB (0, 0, 1600)
```

This statement flushes all open libraries.

Directive Reference

```
*FLUB [ldi]
```

If *ldi* appears in the directive, GMFLUB is called to flush the library connected to that LDI. If omitted, all active libraries are flushed.

§6.4 OPEN LIBRARY: GMOPEN/LMOPEN

This operation opens (activates, assigns, declares) a data library resident on a disk file or main storage, and connects it to a Logical Device Index (LDI). A library must be opened before any I/O activity is attempted on it.

FORTRAN Reference for GMOPEN

Calling sequence

```
CALL GMOPEN (LDI, EDNAME, DDPARS, LBFORM, TRACE)
```

Input Arguments

LDI If LDI > 0, Logical Device Index assigned to a library device. All subsequent references will be done through this LDI. Should this LDI be active, the old device is closed first. This interpretation applies if the "NICE Macroprocessor" flag defined using GMACRO (§10.2) is 0 or 1.

If LDI = 0 on entry, scan the Logical Device Table for an *already active* EDNAME. If found, its LDI is *returned* in this argument (which must therefore be a variable in the calling program), and the open operation skipped. If not found, then search DMGASP's Logical Device Table (ref. 2) for the first *inactive* LDI, set LDI to this value, and continue as in the LDI > 0 case.

If LDI < 0, begin as if LDI = 0, but if an active EDNAME is not found, set LDI = |LDI| and then proceed as in the LDI > 0 case. The absolute value is *returned* in the argument. (Note that if |LDI| happens to be active on entry, the old device will be closed first.)

If the NICE Macroprocessor flag is 2, then LDI > 0 is interpreted as LDI < 0, *i.e.*, all opens are conditional.

EDNAME A character string that contains the *external device name* described in §2.7 of ref. 2. This textstring must be supplied *left-adjusted* and *blank filled.* The name is assumed to be terminated by the *first occurrence of a blank character*, or by the implied length of EDNAME, whichever occurs first. The reader is referred to §§2.7.1-2.7.3 of ref. 2 as regards legal device names for specific computers.

If a blank value is specified for this argument (*i.e.*, EDNAME = ' '), a default name is selected following the rules set forth in Table 2.8 of ref. 2.

DDPARS A four-word integer array that supplies the *device descriptor parameters* discussed in §2.5 of ref. 2.

DDPARS(1) = TYPEX: device type index: an integer in the range 0 through 6. These are listed in Table 6.2, which reproduces Table 2.1 of ref 2. For the distinction between 3 and 4, see Table 2.2 of ref. 2.

DDPARS(2) = OPTX: device options index: an integer in the range -6 through 12. The most useful ones are listed in Table 6.3, which reproduces Table 2.3 of ref. 2.

DDPARS(3) = LIMIT: device capacity limit in words if a new or scratch device. If zero, the default size specified in Table 2.6 of ref. 2 is assumed.

For a core-resident library (TYPEX = 5), LIMIT is the effective blank-common length allocated, starting at the offset prescribed in DDPARS(4).

DDPARS(4) = XPRU for an auxiliary storage device TYPEX < 4, or BCOFF for a core (blank-common-resident) device (TYPEX = 5).

For an auxiliary storage device:

XPRU > 0: external PRU size in words. A value greater than 1 is only useful for DAL files, which are sector-addressable.

XPRU = 0: select XPRU = 1 (word addressing).

XPRU = -1: select XPRU = 1 *and* do Paged I/O to this device if a Page Buffer Pool (PBP) has been previously declared using GMPOOL (§10.7). If no PBP has been declared, XPRU = -1 is the same as 0 or 1.

In summary: For GAL files, select either -1 if you want a Paged I/O device, or 0 if you do not. For DAL files, set XPRU to 28 on Univac, 32 on VAX or IBM, 64 on CDC Cyber.

For a core device, BCOFF is the blank-common offset in words of the device storage allocation. If BCOFF = 0, the device allocation is to start at the first word in blank common. For these devices, XPRU = 1 is implied.

LBFORM A two-word integer array specifying the library format when opening a NEW or SCRATCH device, as set forth in Table 6.4. For additional details, see Table 3.1.

This argument is ignored if opening an existing library, as GAL-DBM can sense the format by scanning its header.

TRACE A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Table 6.2 **The Device Type Index (TYPEX)**

TYPEX	*Device type*
0	Block I/O on disk
1	Reserved for future use
2	Reserved for future use
3,4	FORTRAN direct access
5	"Core device" in blank common

Table 6.3 **The Device Options Index (OPTX)**

OPTX	*Options*
0	Open scratch device
3	Open existing device as read-only
4	Open existing device allowing writes
6	Open new device and catalog as permanent file (public on Univac)
-5	Same as OPTX = 3 if file exists, else same as OPTX = 6
-6	Same as OPTX = 4 if file exists, else same as OPTX = 6

Table 6.4 **The Library Format Array LBFORM**

LBFORM	*Libform*	*Features*
0, 0	GAL80	Holds positional datasets
0, 1	GAL82	Holds nominal datasets
1, 0	DALPRO	Compatible with DALPRO, SPAR

FORTRAN Reference for LMOPEN

LMOPEN is a variant of GMOPEN that has a more human-engineered calling sequence. It is equivalent to a call to GMOPEN with LDI = 0.

Calling Sequence

```
LDI = LMOPEN (KEY, 0, EDNAME, LIMIT, TRACE)
```

Input arguments

KEY	A character string that specifies, in symbolic form, those attributes conveyed by arguments DDPARS and LBFORM in GMOPEN. It consists of a mainkey optionally followed by qualifiers. For example: KEY = 'NEW/DAL '; here NEW is the mainkey while DAL is a qualifier. A list of valid mainkeys and qualifiers appears in Table 6.5.
EDNAME	External device name, as for GMOPEN.
LIMIT	Device capacity limit in words if opening a new or scratch device (same as DDPARS(3) for GMOPEN).
TRACE	Same as for GMOPEN.

Function value

LMOPEN	Returns the assigned LDI if open operation was successful. The procedure followed is identical to that followed by GMOPEN when argument LDI is zero on entry.

EXAMPLE 6.6

Create a GAL80 library to reside on permanent file TEXT*LIBRARY (a Univac file name) and connect it to Logical Device Index 3 using the default LIMIT:

```
INTEGER   DDP(4), LTYP(2)
DATA      DDP /0,6,0,0/, LTYP /0,0/
    :
CALL GMOPEN (3, 'TEXT*LIBRARY ', DDP, LTYP, 2500)
```

Note the blank character terminator after the file name.

Table 6.5 **KEY Argument (Mainkeys and Qualifiers) for LMOPEN**

Mainkey	*Root*	*Effect*
NEW	N	Open new library device and catalog as permanent
OLD	O	Open existing library device allowing writes
ROLD	R	Open existing library device as read-only
SCR	S	Open scratch library device
COLD	C	Open OLD if device exists, else NEW
none		COLD assumed unless BC qualifier given in which case SCR

Qualifier	*Root*	*Valid for mainkeys*	*Effect*
BC	BC	SCR	Blank-common device, with BCOFF = 0
BIO	BI	all	Block I/O device (default if available, else default is F1)
DAL	D	NEW, SCR, COLD	Create DAL libform
F1	F1	all	FORTRAN I/O device with TYPEX=3
F2	F2	all	FORTRAN I/O device with TYPEX=4
GAL80	GAL80	NEW, SCR	Create GAL80 libform
GAL82	GAL82	NEW, SCR	Create GAL82 libform (default)
PIO	P	all but SCR/BC	Paged I/O device (default: Unpaged)

EXAMPLE 6.7

Repeat the above example with **LMOPEN**.

```
LDI = LMOPEN ('NEW/GAL80 ', 0, 'TEXT*LIBRARY ', 0, 2500)
```

In this case the **LDI** is picked by GAL-DBM, and will not necessarily be 3.

Directive Reference

*OPEN [*Qualifiers*] [*ldi*,] *File name* [/LIMIT=*limit*]

This directive is described in detail in ref. 6.

Open Message

When a library file is opened, an informative message is normally written by the I/O Manager to the bulk-print file. For a library resident on auxiliary storage, the message format is typified by the example

```
<DM> OPEN,  LDI: 8, File:  RES.GAL, Attr:  Block I/O, NEW, Paged
```

which is largely self-explanatory. The message above is for a Paged Block I/O device, created on permanent file RES.GAL (a VAX file name) and which will be referenced through LDI number 8. For a FORTRAN I/O device, the logical unit number will be shown before the Attributes text.

The open message for a core-resident library is more concise. For example,

```
<DM> OPEN,  LDI: 12, BC( 30001:  75000)
```

This message says that Logical Device Index 12 will point to a core device that occupies word locations 30001 through 75000 of blank common. No device name is given.

REMARK 6.11

The message is written out *just before* the open request is submitted to either the operating system or the FORTRAN I/O library. Thus, the appearance of the message does not necessarily mean that the operation was successful. If an error condition is detected, a diagnostic will immediately follow (assuming, of course, that the error-file unit is the same as the bulk-print-file unit).

REMARK 6.12

On the Univac version, the message given for *Block I/O devices* has a different format. It is the image of the @ASG request submitted to the Exec-1100 system, followed immediately by the image of the @USE request that links the external and internal file names. For FORTRAN I/O devices, the message has the standard format shown above.

REMARK 6.13

If the case of a conditional open (LDI ≤ 0), no message appears if the operation is skipped because **EDNAME** is already open. Otherwise the message will display the actual LDI chosen by the I/O Manager DMGASP.

REMARK 6.14

Some NICE programmers view these messages as nuisances, especially in highly interactive graphic processors when the bulk-print-file unit is assigned to the screen. The messages may be suppressed (forever or temporarily, as desired), by calling entry point **GMSOCM** (§10.9).

§6.5 PACK LIBRARY: GMPACK

This operation compacts a data library *in situ* by physically removing all deleted datasets. The sequence number of active datasets may change as a result of this operation.

FORTRAN Reference

Calling sequence

```
CALL GMPACK (LDI, 0, TRACE)
```

Input Arguments

LDI	Logical Device Index of library device to be packed.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 6.15

The second argument is presently a dummy parameter.

REMARK 6.16

The present implementation of the pack operation is primitive. A scratch library file is opened, and all active datasets copied to it; then the contents of the original library are replaced by those of the scratch library, which is closed.

EXAMPLE 6.8

Pack library 3 in place.

```
CALL GMPACK (3, 0, 1600)
```

Directive Reference

```
*PACK ldi
```

where *ldi* is the Logical Device index of the library to be packed.

7

Basic Dataset Operations

§7.1 GENERAL DESCRIPTION

This section describes basic operations that affect one or more datasets identified by name or sequence number, and that do not depend on the record infrastructure. Associated entry points are alphabetically ordered by the last four letters of the entry point name (*i.e.*, common roots GM and LM are disregarded). A summary entry point list is provided in Table 7.1.

This Section does not address the following subjects:

1. Operations at the record level (covered in §§8–9).
2. Auxiliary operations such as building dataset names (covered in §10).
3. Information-retrieval entry points (covered in §11).
4. Copy operations (covered in §12).

REMARK 7.1

Any user-program subroutine that references one of the entry points listed in Table 7.1 should first identify itself by calling **GMUSER** as explained in §14.4. This information is used by the central error management routine of NICE-DMS for traceback prints.

Table 7.1 **Entry Points for Basic Dataset Operations**

Operation	*Entry Point*	*Arguments*	*See*
Globally match name	GMATCH	LDI, DSNAME, LOOK, LIST, MLIST, KLIST, TRACE	§7.2
Delete	GMDELD	OPL, LDI, DSNAME, IDSN, TRACE	§7.3
Enable	GMENAD	OPL, LDI, DSNAME, IDSN, TRACE	§7.4
Find first occurrence	LMFIND	LDI, DSNAME, TRACE	§7.5
Find next occurrence	LMFINX	OPL, LDI, DSNAME, IDSNO, TRACE	§7.6
Get name	GMGENT	LDI, DSNAME, IDSN, MR, TRACE	§7.7
Directory	GMLINT GMLIST	LDI, DSNAME, LOPT, TRACE LDI, IDSN1, IDSN2, LOPT, TRACE	§7.8
Set lock code	GMLOCK	LDI, IDSN, LOCK, TRACE	§7.9
Open	GMOPED	OPL, LDI, DSNAME, IDSN, MR, TRACE	§7.10
Put name	GMPUNT LMPUNT	LDI, DSNAME, IDSN, MR, TRACE LDI, DSNAME, TRACE	§7.11
Reserve space	GMREDS	LDI, IDSN, NCHDES, NWRES, TRACE	§7.12
Rename	GMREND	OPL, LDI, DSNAM1, IDSN, DSNAM2, TRACE	§7.13
Set datatype code	GMTYPE	LDI, IDSN, TYPE, TRACE	§7.14

§7.2 GLOBALLY MATCH DATASET NAME: GMATCH

Entry point GMATCH scans the complete Table of Contents of a data library for datasets that match a given name, and returns a list of their sequence numbers. The given name usually has masking and/or cycle-range specifications.

FORTRAN Reference

Calling sequence

```
CALL GMATCH (LDI, DSNAME, LOOK, LIST, MLIST, KLIST, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
DSNAME	Dataset name to be matched. Generally contains mask or cycle-range specifications.
LOOK	Search qualifier: 1 Look for active datasets only. -1 Look for deleted datasets only. 0 Look for active and deleted datasets.
LIST	An integer array dimensioned MLIST or larger, that is to receive the sequence numbers of matched datasets.
MLIST	Maximum number of dataset sequence numbers that may be placed in LIST. Usually equal to the dimension of array LIST in the calling program.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

LIST	Receives KLIST sequence numbers of matched datasets.
KLIST	Count of sequence numbers placed in LIST. (May be zero, but cannot exceed MLIST).

REMARK 7.2

No diagnostic is given if not a single match is made (KLIST = 0).

REMARK 7.3

The Table of Contents is searched from the beginning (sequence number 1). Thus entries placed in array **LIST** will be always in strictly ascending order.

EXAMPLE 7.1

Find all datasets (up to 100) with mainkey **STRUCTURE** in library 3:

```
CALL GMATCH (3, 'STRUCTURE.* ', 0, LIST, 100, KLIST, 2600)
```

EXAMPLE 7.2

Find all *deleted* datasets (up to 200) in library 4:

```
CALL GMATCH (4, '*.* ', -1, LIST, 200, KLIST, 2500)
```

Directive Reference

None.

§7.3 DELETE DATASET: GMDELD

Entry point GMDELD marks one or more datasets as deleted. The dataset specification may be by sequence number or by name. In the latter case, the name argument often contains key-masking or cycle-range specifications.

FORTRAN Reference

Calling sequence

```
CALL GMDELD (OPL, LDI, DSNAME, IDSN, TRACE)
```

Input arguments

OPL	Option letters string. Presently: W Issue a warning diagnostic if a named specification does not match any existing active dataset.
LDI	Logical Device Index of library device.
DSNAME	If nonblank, name that controls the deletion process. If less than 40 characters in length (28 for GAL80 libraries), it should be terminated by a trailing blank. Masking and cycle-range specifications are permitted.
IDSN	If DSNAME is blank, sequence number of dataset to be deleted.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 7.4

GMDELD can delete by sequence and by name.

REMARK 7.5

If option letter W is not given, no warning diagnostic is given if no dataset name is matched by argument DSNAME, or if the lock-code value of any such dataset precludes deletion (see next Remark).

REMARK 7.6

Deletion is precluded if the dataset lock code is 2 or higher (see Table 3.2).

EXAMPLE 7.3

Delete dataset STAR.SHIP in library 14:

```
CALL GMDELD (' ', 14, 'STAR.SHIP ', 0, 1700)
```

EXAMPLE 7.4

Delete from library 14 all datasets whose key extension ends with X, and issue a warning if none exist:

```
CALL GMDELD ('W', 14, '*.*X.* ', 0, 1700)
```

EXAMPLE 7.5

Delete from library 7 all datasets of the form RESPONSE.VECTOR.*cycle1*.0.0, except for the one with highest *cycle1* (this is analogous to VAX/VMS's PURGE command):

```
CALL GMDELD (' ', 7, 'RESPONSE.VECTOR.L:H-1 ', 0, 1700)
```

EXAMPLE 7.6

Delete dataset at sequence 68 in library 11:

```
CALL GMDELD (' ', 11, ' ', 68, 2650)
```

Directive Reference

To delete by sequence range:

*DELETE *ldi*,*idsn1*:*idsn2*

To delete by name:

*DELETE *ldi*,*Dataset_name*

For details, see ref. 5.

EXAMPLE 7.7

Delete all datasets whose mainkey is SOLVE through a message:

```
CALL CLPUT (' *DEL 3,SOLVE.* ')
```

EXAMPLE 7.8

Delete all datasets in sequence range 23 through 44 through a message:

```
CALL CLPUT (' *DEL 3,23:44 ')
```

§7.4 ENABLE DATASET: GMENAD

Entry point GMENAD enables one or more datasets. The dataset identification may be by sequence number or by name. In the latter case, the name may contain masking characters and/or cycle-range specifications.

FORTRAN Reference

Calling sequence

```
CALL GMENAD (OPL, LDI, DSNAME, IDSN, TRACE)
```

Input arguments

OPL	String of option letters. Presently: W Print warning message if a named dataset specification does not match at least one existing deleted dataset.
LDI	Logical Device Index of library device.
DSNAME	If nonblank, name of dataset(s) to be enabled. If less than 40 characters in length (28 for GAL80 libraries), it should be terminated by a trailing blank. Masking and cycle-range specifications are allowed.
IDSN	If DSNAME is blank, sequence number of dataset to be enabled. If DSNAME is nonblank, this argument is ignored.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 7.7

This entry point replaces GMENAB, which was restricted to enabling by sequence number.

REMARK 7.8

No diagnostic is given if the dataset does not exist (sequence number is out of range) or is already enabled.

REMARK 7.9

Before enabling a deleted dataset, GAL-DBM scans the TOC to check whether there is an enabled dataset by the same name. If so, the latter is marked as deleted because names of enabled datasets must be unique. If the lock code of the already-enabled dataset precludes deletion, the operation aborts.

EXAMPLE 7.9

Enable dataset at sequence number 162 in library 12:

```
CALL GMENAD (' ', 12, ' ', 162, 1700)
```

EXAMPLE 7.10

Enable all datasets whose mainkey ends in Z in library 14:

```
CALL GMENAD (' ', 14, '*Z.* ', 0, 1750)
```

Directive Reference

Enable by sequence range:

*ENABLE *ldi,idsn1[:idsn2]*

Enable by name:

*ENABLE *ldi,Dsname*

EXAMPLE 7.11

```
CALL CLPUT (' *ENAB 3,65 ')

CALL CLPUT (' *ENAB 3,COME.BACK ')
```

§7.5 FIND DATASET: LMFIND

LMFIND is the standard entry point for finding an *individual* dataset in a library given its name. If found, the dataset sequence number is returned as the function value.

FORTRAN Reference

Calling sequence

```
IDSN = LMFIND (LDI, DSNAME, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
DSNAME	The name of the dataset to be located. If less than 40 characters in length (28 on GAL80 libraries), it must be terminated by at least one blank. See Remark 7.12 as regards the presence of masking characters or cycle ranges.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Function Return

LMFIND	If the name DSNAME matches that of an active dataset, its sequence number is returned here. Otherwise LMFIND returns zero.

REMARK 7.10

On successful return from this operation for a *positional* dataset, the library file is positioned at the start of record number 1 (*not* at the descriptor start). This facilitates immediate retrieval of the first record. For a *nominal* dataset, the library position is unpredictable.

REMARK 7.11

LMFIND ignores deleted datasets.

REMARK 7.12

The dataset name may contain masking characters or cycle range specifications. If so, the search begins at the TOC start (sequence number 1), and the *first* dataset matched is reported.

REMARK 7.13

If the dataset name does not contain masking characters or cycle range specifications, the TOC search is not necessarily sequential, but is affected by the pattern of paging activity in the TOC Buffer Pool.

EXAMPLE 7.12

Search library connected to LDI = 7 for dataset CONFUCIUS.SAYS.0.0.0 and return its sequence number in IDSN if found:

```
IDSN = LMFIND (7, 'CONFUCIUS.SAYS ', 1300)
```

EXAMPLE 7.13

If no dataset with mainkey SYSTEM is in the library with LDI = 8, jump to subroutine DEFSYS:

```
IF (LMFIND (8, 'SYSTEM.* ', 1400) .EQ. 0)  CALL DEFSYS
```

Directive Reference

*FIND DATASET *ldi*,*Dataset_name* [/SEQ=*Macrosymbol*]

For details, see ref. 5.

EXAMPLE 7.14

```
CALL CLPUT ('*FIND DATASET 7,CONFUCIUS.SAYS /SEQ=IDSN ')
```

§7.6 FIND NEXT DATASET OCCURRENCE: LMFINX

LMFINX is similar to LMFIND (described in the previous subsection), but it commences the search after a specified sequence number. It is normally used to find all datasets that match a masked name, as GMATCH (§7.2) does, but with a return-to-user-program after each match. Additionally, LMFINX can find deleted datasets.

FORTRAN Reference

Calling sequence

```
IDSN = LMFINX (OPL, LDI, DSNAME, IDSNO, TRACE)
```

Input arguments

OPL	Option letters string. Presently D means do not skip deleted datasets.
LDI	Logical Device Index of library device.
DSNAME	The name of the dataset to be located. If less than 40 characters in length (28 on GAL80 libraries), it must be terminated by at least one blank. Generally contains masking and/or cycle-range specifications.
IDSNO	Specifies that TOC search is to begin at sequence number (IDSNO+1).
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Function Return

LMFINX	If name matches that of an active dataset, its sequence number is returned here. Otherwise LMFINX returns zero.

Directive Reference

None.

§7.7 GET NAME FROM TOC: GMGENT

Entry point GMGENT is the inverse of GMPUNT (§7.11). Given the library device LDI and the dataset sequence number, it returns the stored dataset name and a positional/nominal indicator.

FORTRAN Reference

Calling sequence

```
CALL GMGENT (LDI, DSNAME, IDSN, MR, TRACE)
```

Input arguments

LDI — Logical Device Index of library file.

IDSN — Dataset sequence number.

TRACE — A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

DSNAME — A character string that receives the stored name of the IDSNth dataset. The string should have a length of 40 characters (28 for GAL80 libraries) or more in the calling program to avoid potential truncation.

If IDSN is out of range, or some other error condition occurs, DSNAME is blanked.

MR — Returns the value of MR (Maximum Record Access Packets) specified when the dataset name was installed using GMPUNT (§7.11). If the dataset is positional, MR returns zero.

EXAMPLE 7.15

Access dataset at sequence 142 of library 12, and get its name into character string DSN:

```
CHARACTER*40  DSN
    :
CALL GMGENT (12, DSN, 142, M, 2300)
```

Directive Reference

```
*GET DATASET ldi,idsn [/NAME=Macrosymbol]
```

For details, see ref. 5.

§7.8 LIST DATASETS: GMLINT/GMLIST

GMLINT and GMLIST produce a list of datasets stored in the Table of Contents (TOC) of a data library. The list may be limited to datasets that match an input name (GMLINT), or to datasets in a sequence range (GMLIST). Various list formats, controlled by option letters, are provided. Other formats may be added in the future.

FORTRAN Reference for GMLINT

Calling sequence

```
CALL GMLINT (LDI, DSNAME, LOPT, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
DSNAME	Dataset name that specifies the print range. If less than 40 characters in length (28 for GAL80 files), it should be terminated by a blank character. Masking and cycle-range specifications are permitted.
LOPT	An option letter that may be used to control the list format. See Table 7.2.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

FORTRAN Reference for GMLIST

Calling sequence

```
CALL GMLIST (LDI, IDSN1, IDSN2, LOPT, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
IDSN1	If positive nonzero, sequence number at which TOC print is to start. If zero, IDSN1=1 is assumed (first dataset). If negative, list the *last* |IDSN1| datasets in reverse order (IDSN2 is then ignored).
IDSN2	If positive nonzero, sequence number at which TOC print is to stop (if greater than the highest sequence number, print will stop there). If zero, list ends at last dataset in library.

If the effective IDSN2 happens to be less than IDSN1, their role is reversed, and the TOC is listed backward.

LOPT An option letter that may be used to control the list format. See Table 7.2.

TRACE A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 7.14

Deleted datasets that fall in the print range or match **DSNAME** are shown. They are flagged by an asterisk immediately following the sequence number.

EXAMPLE 7.16

Conventional TOC print of all datasets in library 6 whose mainkey starts with **ISM$**:

```
CALL GMLINT (6, 'ISM$*.* ', ' ', 1600)
```

EXAMPLE 7.17

Dated brief TOC print of the highest two first-cycles of datasets named **SHELL.VELOCITY.*.0.0** in library 12:

```
CALL GMLINT (12, 'SHELL.VELOCITY.H-1:H ', 'D', 1800)
```

EXAMPLE 7.18

Conventional listing of the full TOC of library 8:

```
CALL GMLIST (8, 0, 10000, ' ', 2300)
```

EXAMPLE 7.19

SPAR-formatted TOC print of last 16 datasets in library 4:

```
CALL GMLIST (4, -16, 0, 'S', 2400)
```

Directive Reference

*PRINT TOC [/*Form*] *ldi*, *Dataset_name*

where ***PRINT TOC** may be abbreviated to *TOC. For details, see ref. 5.

EXAMPLE 7.20

```
CALL CLPUT ('*TOC 1,RUN.* ')
```

Directive Reference

*PRINT TOC [/*Form*] [*ldi*[,*idsn1* [:*idsn2*]]]

where ***PRINT TOC** may be abbreviated to *TOC. For details see ref. 6.

EXAMPLE 7.21

```
CALL CLPUT ('*TOC 6,-10 ')
```

Table 7.2 **TOC List format Codes**

LOPT *Argument*	*List format*
blank	Conventional: sequence number, date and time of installation, lock code, words stored, record count, generating processor name, type code, dataset name
B	Brief: sequence number and dataset name
D	Dated brief: sequence number, date and time of installation, dataset name
M	DALPRO "Matrix" style (DAL files only)
P	Physical: sequence number, dataset start address, descriptor size, first record size, extent and capacity (in PRUs), dataset name
S	SPAR style (DAL files only)

§7.9 SET LOCK CODE: GMLOCK

Entry point GMLOCK is used to set or change the lock code of a dataset identified by sequence number.

FORTRAN Reference

Calling sequence

```
CALL GMLOCK (LDI, IDSN, LOCK, TRACE)
```

Input arguments

LDI	Logical Device Index of library file.
IDSN	Dataset sequence number.
LOCK	Dataset lock code value. Refer to Table 3.2 for detailed explanation.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 7.15

Dataset locking is not fully implemented.

REMARK 7.16

If GMLOCK is never called, the lock code is zero.

REMARK 7.17

A stored code of 3 or higher can be reduced to 0 or 1 only by the processor that installed the dataset (see Table 3.2 for the details). If this constraint is not met, an error occurs and the operation aborts.

REMARK 7.18

GAL-DBM provides only lock-by-sequence. If you need lock-by-name, write a subroutine using LMFINX.

EXAMPLE 7.22

Make dataset at sequence number 142 read-only.

```
CALL GMLOCK (12, 142, 1, 1900)
```

Directive Reference

None.

§7.10 OPEN DATASET: GMOPED

The open-dataset operation is a "conditional GMPUNT". If the dataset name is already in the library, no operation is performed. Otherwise, the name is installed using GMPUNT.

FORTRAN Reference

Calling sequence

```
CALL GMOPED (OPL, LDI, DSNAME, IDSN, MR, TRACE)
```

Input arguments

OPL	Options letter string. Presently none. Set to blank.
LDI	Logical Device Index of library device.
DSNAME	Name of dataset to be installed. If less than 40 characters in length (28 in GAL80 libraries), it must be terminated by a blank character. Masking or cycle-range specifications are not permitted; if present, an error message will result and the operation aborts. Relative cycle specifications, on the other hand, are allowed (and frequently used).
MR	See GMPUNT
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

IDSN	If dataset already exists, sequence number of existing dataset. If dataset did not exist and the install operation was successful, sequence number assigned to new dataset. If an error occurred, it returns zero.

§7.11 PUT NAME IN TOC: GMPUNT/LMPUNT

Before records can be stored in a new dataset, its name must be installed in the Table of Contents (TOC). This operation is accomplished through entry points GMPUNT and LMPUNT. Note that LMPUNT can only be used to install positional datasets, while GMPUNT can install both positional and nominal datasets.

FORTRAN GMPUNT Reference

Calling sequence

```
CALL GMPUNT (LDI, DSNAME, IDSN, MR, TRACE)
```

Input arguments

LDI — Logical Device Index of library device.

DSNAME — Name of dataset to be installed. If less than 40 characters in length (28 in GAL80 libraries), it must be terminated by a blank character. Masking or cycle-range specifications are not permitted; if present, an error message will result and the operation aborts. Relative cycle specifications, on the other hand, are allowed (and frequently used).

MR — This argument is relevant only for nominal datasets, in which case

MR > 0: MR is an upper bound on the number of Record Access Table entries to be used. See Remark 7.19.

If installing a positional dataset (GAL80 library), set MR to zero.

TRACE — A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

IDSN — If the operation is successful, sequence number of new dataset, otherwise, it is zero.

REMARK 7.19

To come up with a value for MR, do the following. Estimate (1) number of ordinary records to be stored, and (2) number of record groups to be stored, add the two together and round up to the next 16-multiple. As discussed in the next remark, the estimate can be very coarse.

REMARK 7.20

The value of MR is used to size up the RAT Header at one word for each 16 RAT entries. Accordingly, a coarse estimation is sufficient. For example, going from MR = 160 to 320 makes a difference of 10 words, which is not very significant.

FORTRAN LMPUNT Reference

Function Reference

```
IDSN = LMPUNT (LDI, DSNAME, TRACE)
```

Input arguments

The input arguments have the same meaning as for GMPUNT. Note the absence of the MR argument precludes the installation of nominal datasets with LMPUNT.

EXAMPLE 7.23

Install positional dataset **CONFUCIUS.SAYS** in library 7:

```
CALL GMPUNT (7, 'CONFUCIUS.SAYS ', IDSN, 0, 1300)
```

EXAMPLE 7.24

Install dataset **RESPONSE.VEC.N** as nominal dataset name in library 11 with an MR estimate of 48 RAT entries:

```
CALL GMPUNT (11, 'RESPONSE.VEC.N ', IDSN, 48, 1600)
```

Directive Reference

*PUT DATASET *ldi*,*Dataset_name* [/MR=*mrat*] [/SEQ=*Macrosymbol*]

For details, see ref. 5.

EXAMPLE 7.25

```
CALL CLPUT ('*PUT DATA 1,NEW.START.N ')
```

§7.12 RESERVE DATASET SPACE: GMREDS

Entry point GMREDS is used to force GAL-DBM into reserving physical space for a *positional* dataset just installed using GMPUNT or LMPUNT. The operation has no meaning for nominal datasets.

FORTRAN Reference

Calling sequence

```
CALL GMREDS (LDI, IDSN, NCHDES, NWRES, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number returned by GMPUNT or LMPUNT.
NCHDES	Number of characters to be reserved for the descriptor record at the dataset start. This space, rounded up to the next "covering" word, is filled with blanks. If NCHDES = 0, no descriptor reservation is made.
NWRES	Number of words to be reserved for dataset proper. A negative value requests that the |NWRES| space be physically filled with zero words. If NWRES = 0, no space is reserved.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

NCHDES	If GMREDS is not successful, NCHDES is set to zero.
NWRES	If GMREDS is not successful, NWRES is set to zero.

REMARK 7.21

This operation is confined to positional datasets. If it is tried on a nominal library, the operation is ignored.

REMARK 7.22

GMREDS replaces function entry point **LMPURS**, described in the previous version of this document. **LMPURS** combined the functions of **LMPUNT** and **GMREDS**.

EXAMPLE 7.26

Install positional dataset **LOW.PROFILE.6** in library 11, and reserve 240 characters for the descriptor and **52000 words** for the dataset proper.

```
CALL GMPUNT (11, 'LOW.PROFILE.6 ', IDSN, 0, 1300)
CALL GMREDS (11, IDSN, 240, 52000, 1400)
```

Directive Reference

None.

§7.13 RENAME DATASET: GMREND

Entry point GMREND changes the name of one or more datasets. The dataset specification may be by sequence number or by name.

FORTRAN Reference

Calling sequence

```
CALL GMREND (OPL, LDI, DSNAM1, IDSN, DSNAM2, TRACE)
```

Input arguments

OPL	Options letter string. Presently W = print warning if DSNAM1 is non-blank and there are no matches.
LDI	Logical Device Index of library file.
DSNAM1	If non-blank, dataset name(s) to be renamed. Ignored if blank. If less than 40 characters in length (28 for GAL80 libraries), it must be terminated by a blank character.
IDSN	If DSNAM1 is blank, sequence number of dataset whose name is to be changed. Else ignored.
DSNAM2	New dataset name. If less than 40 characters in length (28 for GAL80 libraries) it must be terminated by a blank character. Fully masked components allowed.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 7.23

Before the name is changed, GAL-DBM scans the entire TOC for matches. Any active dataset whose name matches that of the new name argument is deleted.

EXAMPLE 7.27

Rename dataset 68 in library 14 to PHOENIX.ASH.N:

```
CALL GMREND (' ', 14, ' ', 68, 'PHOENIX.ASH.N ', 1900)
```

Directive Reference

```
*RENAME DATASET ldi,idsn = Dataset_name
```

For details, see ref. 5.

§7.14 SET DAL-DATATYPE CODE: GMTYPE

Entry point **GMTYPE** is used to set the datatype code of a DAL-conforming dataset identified by sequence number.

FORTRAN Reference

Calling sequence

```
CALL GMTYPE (LDI, IDSN, TYPE, TRACE)
```

Input arguments

LDI	Logical Device Index of library file.
IDSN	Dataset sequence number.
TYPE	DAL-datatype code value (Table 4.1).
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 7.24

Before **GMTYPE** is called, the datatype code of a *positional dataset* is zero. A nominal dataset does not have a type (its records do).

REMARK 7.25

These datatype codes are only useful for DAL-conforming datasets (§4.2.1). For other positional datasets (*e.g.*, GAL-conforming) a nonzero type code may be stored through **GMTYPE**, but it is innocuous.

REMARK 7.26

If the **IDSN**th dataset is of nominal type, this operation is ignored.

EXAMPLE 7.28

Identify dataset at sequence number 81 of library 9 as a variable-length-image text dataset (DAL-PRO type 5):

```
CALL GMTYPE (9, 81, 5, 2300)
```

Directive Reference

None.

8

Indexed Record Operations

§8.1 GENERAL DESCRIPTION

This section covers operations for handling indexed records resident in positional datasets. These operations are illegal or meaningless if tried on nominal datasets.

Presentation of indexed-record operations is alphabetically ordered by the last four letters of the entry point name (*i.e.*, common roots GM and LM are disregarded). A summary entry point list is provided in Table 8.1.

REMARK 8.1

Any user-program subroutine that references one of the following entry points should first identify itself by calling **GMUSER** as explained in §14.4. This information is used by the central error management routine of NICE-DMS for traceback prints.

Table 8.1. **Entry Points for Indexed-Record Operations**

Operation	*Entry Point*	*Arguments*	*See*
Find end	GMFEND	LDI, IDSN, TRACE	§8.2
	LMFEND	LDI, IDSN, TRACE	
Find record	GMFIRE	LDI, IDSN, IREC, IOFF, TRACE	§8.3
	LMFIRE	LDI, IDSN, IREC, IOFF, TRACE	
Position and read characters	GMPORC	LDI, IDSN, IREC, A, N, IOFF, TRACE	§8.4
	LMPORC	LDI, IDSN, IREC, A, N, IOFF, TRACE	
Position and read numerics	GMPORN	LDI, IDSN, IREC, A, N, IOFF, TRACE	§8.4
	LMPORN	LDI, IDSN, IREC, A, N, IOFF, TRACE	
Position and write characters	GMPOWC	LDI, IDSN, IREC, A, N, IOFF, TRACE	§8.4
	LMPOWC	LDI, IDSN, IREC, A, N, IOFF, TRACE	
Position and write numerics	GMPOWN	LDI, IDSN, IREC, A, N, IOFF, TRACE	§8.4
	LMPOWN	LDI, IDSN, IREC, A, N, IOFF, TRACE	
Print records	GMSHOP	OP, LDI, IDSN, IREC1, IREC2, PFORM, M, IOFF, TRACE	§8.5
Transfer characters	GMTRAC	OP, LDI, A, N, TRACE	§8.6
Transfer numerics	GMTRAN	OP, LDI, A, N, TRACE	§8.6

§8.2 FIND DATASET END: GMFEND/LMFEND

Two entry points: GMFEND and LMFEND may be used to *position* the library device to the *end* of an indexed-record dataset. This operation is normally done as a prelude to an append-record operation using GMTRAN or GMTRAC.

GMFEND is referenced as a subroutine whereas LMFEND is referenced as an integer function. Both take as inputs the Logical Device Index and dataset sequence number. LMFEND returns, as function value, the number of words that can be appended to the dataset.

FORTRAN GMFEND Reference

Calling sequence

```
CALL GMFEND (LDI, IDSN, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 8.2

On error-free return from this operation, the IDSNth dataset is the active dataset, and the library device is positioned at the dataset end.

REMARK 8.3

This operation works correctly for all types of indexed-record datasets, whether GAL-conforming or not.

REMARK 8.4

This operation, as well as LMFEND below, is illegal for nominal datasets, since the term "dataset end" is then meaningless.

REMARK 8.5

The main use of GMFEND is to make the IDSNth dataset the active one.

FORTRAN LMFEND Reference

LMFEND performs exactly the same job as **GMFEND**, but is referenced as an integer function:

```
MORE = LMFEND (LDI, IDSN, TRACE)
```

Input arguments

The three input arguments are identical to those of GMFEND.

Function return

LMFEND Number of *words* that can be appended to the dataset.
A zero return value means one of three things:
(a) the dataset cannot be expanded;
(b) the dataset is locked as read-only (Table 3.2); or
(c) an error was detected.

REMARK 8.6

LMFEND is generally preferable to **GMFEND**, as the example below makes plain. **GMFEND** should be used only if you are sure, beyond a reasonable doubt, that nothing can go wrong with the subsequent record append.

EXAMPLE 8.1

Append a 250-word record in array B to dataset **AAA.BBB.3.0.0** of library 6 if there is enough room to do so. If there is not enough room, move the dataset to the end of the data library and append the record.

```
1200  IDSN = LMFIND (6, 'AAA.BBB ', 1300)
      IF (IDSN .GT. 0)     THEN
        IF (LMFEND (6, IDSN, 1700) .LT. 250)   THEN
           CALL GMCOPS (6, IDSN, 6, ...  )
           IF (LMERCD(IERR) .NE. 0) CALL ERROR (IERR)
           GO TO 1200
        END IF
        CALL GMTRAN ('W/A', 6, B, 250, 1500)
      END IF
      :
```

§8.3 FIND INDEXED RECORD: GMFIRE/LMFIRE

GMFIRE and LMFIRE are used to *position* the library device to the start of an indexed record. GMFIRE is called as a subroutine while LMFIRE is referenced as an integer function.

The inputs are the Logical Device Index, dataset sequence number, record index and offset from record start. If the record exists, its size is returned by LMFIRE as function value.

This operation is guaranteed to work correctly only for DAL- and GAL-conforming positional datasets. (see Remark 8.9 for more details).

FORTRAN GMFIRE Reference

Calling sequence

```
CALL GMFIRE (LDI, IDSN, IREC, IOFF, TRACE)
```

Input arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number. A zero implies the active dataset.
IREC	The record index. IREC = 0 calls for the descriptor record. See Remark 8.10 for the effect of an out-of-bounds IREC.
IOFF	Record-offset argument: IOFF = 0 positions to record start. If nonzero, position GAL library at IOFF *words* from the record start. For restrictions on the use of nonzero IOFF, see Remarks 8.11.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 8.7

This operation is illegal for nominal datasets.

REMARK 8.8

On successful return from this operation, the library device is positioned at record start if IOFF = 0, otherwise, at IOFF words from it.

REMARK 8.9

This operation works correctly only if record lengths meet the constraints outlined in §4.2 for GAL-conforming datasets (which include DAL-conforming datasets as a special case). For more general record-length distributions (the so-called "wild datasets"), GMFIRE will position correctly to records 0, 1, 2, 3, last and next-to-last, but not necessarily otherwise. As GAL-DBM tacitly assumes that all positional datasets are GAL-conforming, incorrect positioning is undetectable.

REMARK 8.10

If the record index is too large (exceeds number of stored records), the library is position to the *dataset end*, just like **GMFEND** (§8.2). **IOFF** is then ignored.

REMARK 8.11

A nonzero **IOFF** should not be used for character records, because the results will generally be unpredictable. If you must offset, consider using **GMPORC/GMPOWC** (§8.4).

REMARK 8.12

A nonzero **IOFF** should not be used for DAL libraries, because these have sector addressability and cannot "remember" word offsets.

FORTRAN LMFIRE Reference

LMFIRE performs exactly the same job as **GMFIRE**, but is referenced as an integer function:

```
SIZE = LMFIRE (LDI, IDSN, IREC, IOFF, TRACE)
```

The five arguments have the same meaning as for **GMFIRE**. The function output is:

LMFIRE	Zero if record is not in dataset, or an error was detected. Otherwise size of record. Size is given in *words* if **IREC** $\geq$ 1, or in characters if **IREC** is zero.

REMARK 8.13

The function form is handy when the record size return is only to be used in an **IF** statement for testing the presence or absence of a record, as in the example below.

EXAMPLE 8.2

If dataset **AAA.BBB.0.0.0** is in library 7, read its second record into array B:

```
IDSN = LMFIND (4, 'AAA.BBB ', IDSN, 1300)
IF (IDSN .GT. 0)     THEN
  N = LMFIRE (4, IDSN, 2, 0, 1700)
  IF (N .GT. 0)    THEN
    CALL GMTRAN ('R', 4, B, N, 1500)
  END IF
END IF
```

EXAMPLE 8.3

If dataset **STIFFNESS.MATRIX** in library 3 has no second record, write a 60-word zero-filled record in the second record.

```
IDSN = LMFIND (4, 'STIFFNESS.MATRIX ', IDSN, 1300)
IF (IDSN .GT. 0)      THEN
  IF ( LMFIRE (4, IDSN, 2, 0, 1700) .EQ. 0)    THEN
    CALL GMTRAN ('W/F', 4, 0, 60, 1500)
  END IF
END IF
```

§8.4 POSITION AND READ/WRITE: GMPO*xx*/LMPO*xx*

Entry points GMPO*xx* and LMPO*xx*, where *xx* = RC, RN, WC or WN, combine library positioning and physical data transfer in a single operation. They can be used to *read* or *rewrite* existing records or *segments* of existing records. They cannot be used to *create* new records, or to *extend* the last record, a job which is reserved to GMTRAN/GMTRAC (§8.6).

The eight entry points are collectively described here, as they share the same calling sequence. The dataset record, or segment thereof, is defined by Logical Device Index, dataset sequence number, record index, and offset from record start. The main-storage array is defined by its address and length. Record lengths and offsets are reckoned in words for numerical records and characters for character records.

These operations are legal only on *positional datasets* resident in *word-addressable* GAL devices. They are illegal on DAL files, or on nominal datasets.

FORTRAN GMPO*xx* References

The calling sequence for the four entry points is:

```
CALL GMPOxx (LDI, IDSN, IREC, A, N, IOFF, TRACE)
```

The entry-point name combinations are:

GMPORC	Position and read character data.
GMPORN	Position and read numeric data.
GMPOWC	Position and write character data.
GMPOWN	Position and write numeric data.

Input Arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.
IREC	Record index.
A	*xx* = RC: character array that receives record. *xx* = RN: numeric array that receives record. *xx* = WC: character array to be stored. *xx* = WN: numeric array to be stored.
N	The absolute value of N is the maximum number of characters (*xx* = RC or WC) or words (*xx* = RN or WN) to be read (*xx* = RC or RN) or written (*xx* = WC or WN). A negative value of N assures that the read or write will not extend beyond the boundary of the dataset. A positive

value of N assures that the read or write will not extend beyond the boundary of the IRECth record. A value of zero for N will cause GMPORN and GMPOWN to position only, and GMPORC and GMPOWC to use the implicit length of A for N.

If the calling program needs to find out how many characters or words have been actually transferred, the function entry points LMPO*xx* discussed below should be used instead.

IOFF *xx* = RC: offset in characters at which read is to begin.
xx = RN: offset in words at which read is to begin.
xx = WC: offset in characters at which write is to begin.
xx = WN: offset in words at which write is to begin.

TRACE A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 8.14

If the library device is of DAL form, or the IDSNth dataset is nominal, an error exit is taken.

REMARK 8.15

On exit from any of these operations, the library position is unpredictable.

REMARK 8.16

If the record does not exist, or if IOFF puts the read or write totally out-of-bounds, no data transfer occurs. This condition is not reported as an error condition.

FORTRAN LMPO*xx* References

The four entry points discussed in this section may also be invoked as integer functions:

```
NX = LMPOxx (LDI, IDSN, IREC, A, N, IOFF, TRACE)
```

where *xx* is again RC, RN, WC or WN. The seven arguments have identical meaning. The function value returns the number of characters or words actually transferred. This value may be zero if an error occurs or if the read/write is totally out of dataset bounds.

REMARK 8.17

If the record does not exist, or if IOFF puts the read or write totally out-of-bounds, no data transfer occurs. The function returns zero.

§8.5 PRINT RECORDS: GMSHOP

GMSHOP provides a type-directed display of the contents of positional dataset records.

FORTRAN GMSHOP Reference

Calling sequence

```
CALL GMSHOP (OP, LDI, IDSN, IREC1, IREC2, PFORM, M, IOFF, TRACE)
```

Input arguments

OP	A character string that specifies print options. D Place dataset name in label. R Place record ID in label. X Suppress title line. V Force TTY print. W Give warning if no records found.
LDI	Logical device index of library file.
IDSN	Dataset sequence number.
IREC1	Index of first record to be printed (≥ 0)
IREC2	Index of last record to be printed (≥ 0)
PFORM	Print format specification. If blank, GMSHOP will try to think of something.
M	If M > 0, limit record print to M items.
IOFF	Offset to first printed item. Applies to all records. Normally zero.
TRACE	Error traceback argument. Do not use a zero or negative value here; these values are reserved for internal use.

EXAMPLE 8.4

Print the first 3 records of dataset number 6 on library 10.

```
CALL GMSHOP('DR',10,6,1,3,' ',0,0,100)
```

§8.6 TRANSFER DATA: GMTRAx

GMTRAx, where x = C or N, are standard entry points for reading or writing indexed records. GMTRAC is used for character records whereas GMTRAN is used for numeric records. Both require appropriate *prepositioning* of the library device to the location at which the read or write is to take place.

GMTRAx consolidates the functions of a number of entry points treated in the previous version of this document: GMREAD, GMREAC, GMWRIN, GMWRIB, GMWRIT, GMWRIC, GMWRIZ, LMRDES and LMWDES.

For the current GAL-DBM version, GMTRAC and GMTRAN provide the *only* way to perform the following operations: (1) appending indexed records to GAL or DAL datasets; (2) reading, writing or rewriting DAL records.

FORTRAN GMTRAx Reference

Calling sequence

```
CALL GMTRAx (OP, LDI, A, N, TRACE)
```

The entry-point name combinations are:

GMTRAC	Transfer character data.
GMTRAN	Transfer numeric data.

Input arguments

OP — A character string that specifies the operation to be performed. It consists of a mainkey optionally followed by qualifiers. For example:

```
OP = 'WRITE/FILL'
```

Here WRITE is the mainkey whereas FILL is a qualifier. A list of valid mainkeys and qualifiers appears in Table 8.2.

Mainkeys and qualifiers may be abbreviated to the "roots" shown in Table 8.2.

LDI — Logical Device Index of library device.

A — x = C, OP = 'R': character string that receives record.

x = C, OP = 'R/D': character string that receives descriptor record.

x = C, OP = 'W/A': character string to be appended as new record.

x = C, OP = 'W/D': character string to be written to descriptor record.

x = C, OP = 'W/F': A(1:1) is fill character.

x = C, OP = 'W/U': character string to be rewritten at current location.

x = N, OP = 'R': numeric array that receives record.

x = N, OP = 'W/A': numeric array to be appended as new record.

x = N, OP = 'W/F': A(1) is fill word.

x = N, OP = 'W/U': numeric array to be rewritten at current location.

x = N, OP = 'W/X': numeric array to be appended as last-record extension.

N — x = C: number of characters to be read or written. If N = 0, **LEN(A)** is used.

x = N: number of words to be read or written.

TRACE — A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Library Prepositioning

Unlike the GMPOxx entry points (§8.4), the calling sequence of GMTRAx contains no dataset or record identifiers. These items are *implicitly* defined by prior operations on the target data library, and by the OP argument.

It follows that references to GMTRAC or GMTRAN cannot occur in isolation, but must be appropriately prepared. The preparation procedure is called *library prepositioning.* Table 8.3 lists recommended prepositioning procedures corresponding to common OP settings. These procedures are safe in the sense that it is sometimes possible to take shortcuts and get away with it, but subsequent modifications in the user program may cause problems.

REMARK 8.18

The omission of dataset and record identifiers from GMTRAx is not, by the way, accidental. The resulting low processing overhead makes these routines very efficient for *sequentially* reading or writing indexed record streams. This situation is precisely the kind of situation in which positional datasets hold a definite edge over nominal datasets.

EXAMPLE 8.5

See Examples in §§8.2–8.3.

Table 8.2. **OP Argument (Mainkeys and Qualifiers) for GMTRA*x***

Mainkey	*Root*	*Effect*
READ	R	Read; see qualifiers for details
WRITE	W	Write; see qualifiers for details
none		READ assumed

Qualifier	*Root*	*Valid for mainkeys*	*Effect*
APPEND	A	W	Append new indexed record
DESCRIPTOR	D	R,W	Read or write descriptor; GMTRAC only
FILL	F	W	Fill with A(1)
UPDATE	U	W	Rewrite at current location
XTEND	X	W	Extend last record; GMTRAN only
none		R	Read at current location
none		W	W/A if library is positioned at end of dataset; else W/U

Table 8.3. **Library Prepositioning for GMTRAx**

OP *Argument*	*Prepositioning*
R	Call LMFIRE or GMFIRE to set target record and offset. Successive calls to GMTRAx to read *consecutive* records, or segments of one record, are permitted.
R/D	Call LMFIRE or GMFIRE with IREC=IOFF=0.
W/A or W/A/F	Call LMFEND or GMFEND to position to dataset end. (This may be dispensed with if (a) this is record #1 and (b) the previous GAL-DBM reference is a call to GMPUNT (or GMPUNT/GMREDS) to install dataset.) Successive calls to GMTRAx to write-append *consecutive* dataset records are permitted.
W/D	Call LMFIRE or GMFIRE with IREC=IOFF=0. (This may be dispensed with if the previous reference to GAL-DBM is a call to GMPUNT (or GMPUNT/GMREDS) to install dataset.)
W/U or W/U/F	Same as for R.
W/X or W/X/F	Essentially the same as for W/A, but more complicated. Only experienced users should use the X qualifier.

9

Named Record Operations

§9.1 GENERAL DESCRIPTION

This section covers operations on named records. The generic operations are *get* and *put*, which together take care of initialization, space reservation, retrieval, seeking, storage and updating of named records.

Complementing the basic get/put operations are information retrieval entry points to get attributes such as record group high/low cycles, etc, print RAT, print records, delete and rename records. Nominal-dataset display operations are: list Record Access Table and print record contents. The record or record(s) subject of these operations are identified by name; consequently, these operations are illegal on positional datasets.

The presentation of named-record operations in this Section is alphabetically ordered. A summary entry point list is provided in Table 9.1.

REMARK 9.1

Any user-program subroutine that references one of the following entry points should first identify itself by calling **GMUSER** as explained in §14.4. This information is used by the central error management routine of NICE-DMS for traceback prints.

Table 9.1 **Entry Points for Named-Record Operations**

Operation	*Entry Point*	*Arguments*	*See*
Delete record(s)	GMDERT	OP, LDI, IDSN, RNAME, TRACE	§9.2
Get Group cycles	GMGECY	OP, LDI, IDSN, RKEY, NREC, IL, IH, TRACE	§9.3
Get Key Attributes	GMGEKA	OP, LDI, IDSN, RKEY, RTYPE, N, M, TRACE	§9.4
Get record(s)	GMGET*x*	OP, LDI, IDSN, RNAME, TYPE, A, N, M, IGAP, IOFF, TRACE	§9.5
List RAT	GMLIRT	OP, LDI, IDSN, RKEY, LISTYP, TRACE	§9.6
Print record(s)	GMSHOR	OP, LDI, IDSN, RNAME, PFORM, M, IOFF, TRACE	§9.7
Rename record(s)	GMRERT	OP, LDI, IDSN, OLDKEY, NEWKEY, TRACE	§9.8
Put record(s)	GMPUT*x*	OP, LDI, IDSN, RNAME, TYPE, A, N, M, IGAP, IOFF, TRACE	§9.9

§9.2 DELETE RECORD(S): GMDERT

Entry point GMDERT marks one or more named records as pertaining to a dataset as deleted. Deletion is performed by removal or modification of packets of the Record Access Table (RAT) of the dataset. If the operation results in the removal of a packet, a RAT compression may ensue.

Unlike datasets, deleted records cannot be enabled back to an active status. Space occupied by deleted records can be reclaimed on a pack operation.

FORTRAN Reference

Calling sequence

```
CALL GMDERT (OP, LDI, IDSN, RNAME, TRACE)
```

Input Arguments

OP A character string containing operation specifications. Presently:

K Delete by key. Argument RNAME is then a record key, and all records having that key are deleted. Cycles disregarded.

P Compress RAT if record deletion results in one or more RAT packets being vacated.

W Print warning if nothing deleted.

LDI Logical Device Index of library device.

IDSN Sequence number of owner dataset.

RNAME A name that identifies the record(s) to be deleted. The specification takes different forms according to whether option K is specified in argument OP. If K is not specified, the general form is

$$Key.n_1{:}n_2$$

and all records that intersect with this specification are deleted. An error condition may occur if one tries to delete intermediate cycles from a record group (see Remark 9.2). Masking specifications on the record key are acceptable if the cycles are explicitly given. If the K option is given in argument OP, only the key needs to be given. Specifying RNAME = '*' in conjunction with OP = 'K' results in all records being deleted.

TRACE A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 9.2

Deleting cycles from a record group may be a risky business. For example, suppose that record group `Z.10:45` uses a RAT packet, and consider the effect of requesting the deletion of

```
Z.1:25
Z.35:50
Z.20:30
```

The first two specifications are legal. Upon return from `GMDERT`, the record group is truncated to `Z.26:45` and `Z.10:34`, respectively. But the last request is illegal because it would result in a *split* record group, which cannot be managed from a single packet, and an error condition will be reported.

§9.3 GET GROUP CYCLES: GMGECY

Entry point GMGECY returns the lowest and highest defined cycles of a record group, given the record key. The record group may be unsegmented or segmented.

FORTRAN Reference

Calling sequence

```
CALL GMGECY (OP, LDI, IDSN, RKEY, NREC, IL, IH, TRACE)
```

Input Arguments

OP	A character string containing operation specifications. Not presently used.
LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.
RKEY	Record Group key.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

NREC	Number of records found. If key is undefined, NREC = 0.
IL, IH	If NREC > 0, low and high record cycles, respectively. For an unsegmented record group, NREC = IH−IL+1 is guaranteed; not so, however, for a segmented one if cycle gaps occur. If NREC = 0, IL = IH = −1.

EXAMPLE 9.1

Retrieve low and high cycles of unsegmented record group keyed NODES, located in dataset GEOMETRIC.TABLES of library 7:

```
IDSN = LMFIND (7, 'GEOMETRIC.TABLES ', 1600)
CALL GMGECY (' ', 7, IDSN, 'NODES ', KREC, ILO, IHI, 1700)
```

§9.4 GET KEY ATTRIBUTES: GMGEKA

GMGEKA is a "key attribute" retriever. It is given as input a record key and the library and dataset in which it resides. It returns as output the attributes that are invariant across all records with that key, namely data type, item length, and first dimension value.

FORTRAN Reference

Calling sequence

```
CALL GMGEKA (OP, LDI, IDSN, RKEY, RTYPE, N, M, TRACE)
```

Input Arguments

OP	A character string containing operation specifications. Presently this is a dummy argument, so it should be set to ' '.
LDI	Logical Device Index of library device.
IDSN	Dataset sequence number. A zero implies the active dataset.
RKEY	A character string containing the record key left-justified. If the number of characters is less than the maximum key length, it should be terminated by blank-fill.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

RTYPE	If records identified by RKEY exist, then their data type is RTYPE (as defined in Table 5.1). Otherwise blank.
N	If records identified by RKEY exist, then their logical length is N. Otherwise zero.
M	If records identified by RKEY exist, then their first matrix dimension attribute is M. Otherwise zero.

EXAMPLE 9.2

Retrieve attributes of record key VELOCITIES in dataset at sequence number 181 in library connected to LDI 15:

```
CALL GMGEKA (' ', 15, 181, 'VELOCITIES ', RT, L, M, 1700)
```

where RT is a CHARACTER*1 variable.

§9.5 GET NAMED RECORD(S): GMGET*x*

GMGETN (get numerics) and GMGETC (get characters) are the standard entry points for accessing *existing* named records. "Accessing" means reading or finding. *Reading* means that stored-record data is transferred into a main storage array, whereas *finding* implies that the dataset is searched to acquire length and type information albeit no data transfer occurs. The operation may involve a single record, a record group, or a table.

The two entry points are collectively described here as they share the same calling sequence. An operation key specifies various options. The database record(s) are described by the Logical Device Index, dataset sequence number, and a record name; the latter may involve key concatenation and cycle ranges. If records are to be read, a destination main storage array is specified, as well as variables to receive record size and type. Size constraints may be specified on input.

FORTRAN Reference

Calling sequence

```
CALL GMGETx (OP, LDI, IDSN, RNAME, TYPE, A, N, M, IGAP, IOFF, TRACE)
```

The entry-point names are:

GMGETC	Get character record(s).
GMGETN	Get numeric record(s).

Input Arguments

OP A character string that specifies the operation to be performed. It consists of a mainkey optionally followed by one or more qualifiers. For example:

```
OP = 'READ/LENGTH'
```

Here READ is the mainkey while LENGTH is a qualifier. There can be only one mainkey but several qualifiers (or none). A list of valid mainkeys and qualifiers appears in Table 9.2. Mainkeys and qualifiers may be abbreviated to the one-letter "roots" shown there.

LDI Logical Device Index of library device.

IDSN Dataset sequence number. A zero implies the active dataset.

RNAME A character string containing the name of the record, record group, or table to be accessed. This string should be terminated by a blank character for safety.

What if no records by this name are found? See Remark 9.4.

TYPE If OP mainkey is READ, external datatype code (Table 5.1) of variable or array that will receive data (the argument A). Generally this type should

match that of the record(s) you want to retrieve. However, GMGETN will do certain numeric conversions for you; the ones presently allowed are shown in Table 9.3. If the datatype mismatch is not one of these, the operation aborts. See remark 9.6 below for TYPE = ' '

If OP mainkey is FIND, TYPE is an output argument (see below).

A — If OP mainkey is READ, array that will receive record, Record Group, or Table. For GMGETC, A must be a character string or character array. For GMGETN, A must be a numeric array of the type specified by TYPE. For multi-record read, see IGAP below.

If OP mainkey is FIND, A is a dummy argument. If you are calling GMGETN, put a zero here. If you are calling GMGETC, put a blank character here.

N — If OP is READ and LENGTH appears as qualifier, N is an input-output argument. The input value of N may be positive or negative.

An input N > 0 tells GMGETx: "do not read more than N items". This limit usually reflects the main-storage allocation of A in the calling program and is specified as a safety factor against array overspill.

An input N < 0 tells GMGETx: "do not read more than |N| items per record". This has some applications when reading record groups or tables; see example. For individual record retrieval, +N and -N are equivalent.

If LENGTH is not a qualifier, N is only an output argument.

M — A dummy integer argument unless qualifier MATRIX appears, in which case M is an output argument (see below).

IGAP — If OP qualifier is READ, this argument is applicable to multi-record transfer as follows: skip IGAP items in array A when incrementing the *cycle* number; but leave no gap if IGAP = 0. See examples.

IOFF — If OP mainkey is READ: begin record read at IOFF items from record start.

If OP mainkey is FIND, IOFF is a dummy integer argument.

TRACE — A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

Output Arguments

TYPE — If OP mainkey is FIND, external datatype code (Table 5.1) of stored record(s). (Should be declared CHARACTER*1 in calling program.)

If RNAME specifies a Table, and records are found to be of different types, TYPE = 'M' (Mixed) is returned.

A	If OP mainkey is read, A receives record data found by GMGETC or GMGETN. The configuration of A upon multi-record read (Record Group or Table) is illustrated by Examples 9.3 through 9.12.
N	If OP mainkey is READ, N returns *total* number of items read into A. If OP mainkey is FIND, N returns total number of items found. If no record is found, or an error is detected before the record search starts, N returns zero.
M	If qualifier MATRIX appears, M returns the "first matrix dimension" stored by GMPUT*x*.

REMARK 9.3

The use of mainkey FIND is admittedly rare now that information-retrieval entry points GMGEKA and GMGECY are available. These entry points are not so comprehensive but have a simpler calling sequence.

REMARK 9.4

GMGET*x* produces no diagnostics if a record specified by RNAME is not found, or even if nothing is found. The user program may check the output N to make sure that the requested data is there. For example, suppose that RNAME = 'QUERCUS.1:60 ', where each record is 10-items long, but only QUERCUS.3:6 are actually stored in the dataset. Only 4 records will be read, and the exit N = $10 \times 4 = 40$.

REMARK 9.5

If it is important to determine in advance whether a record or record group exists, call GMGET*x* with OP = 'FIND', To find out the cycle range of a record group, use GMGECY (§9.3). See example for a realistic application of these "query" routines.

REMARK 9.6

If TYPE is declared unknown (U) on a GMGETN read operation, numeric records of *any* datatype will be moved to A without conversion. (This mimics the *modus operandi* of positional datasets.) But this is not allowed for character records; these have to be read using GMGETC, and the input datatype must be A.

Table 9.2. **OP Argument (Mainkeys and Qualifiers) for GMGETx**

Mainkey	*Root*	*Effect*
FIND	F	Find record(s) and return information.
READ	R	Read records into argument array.
none		READ assumed.

Qualifier	*Root*	*Valid for mainkeys*	*Effect*
LENGTH	L	READ	N is an input-output argument
MATRIX	M	READ,FIND	Return first matrix dimension in M.

Table 9.3. **Mixed Datatype Handling by GMGETx**

Argument	Database: *Integer*	*S-float*	*D-float*	*Complex*	*Character*
Integer	Yes	No	No	No	No
S-Float	No	Yes	Yes	No	No
D-Float	No	Yes	Yes	No	No
Complex	No	No	No	Yes	No
Character	No	No	No	No	Yes
Unknown	Yes	Yes	Yes	Yes	No

EXAMPLE 9.3

To illustrate the use of GMGETN and GMGETC, the following Table-compatible four record groups are assumed to be present in dataset 55 of library 7:

S	J	XYZ	ABCD
s_1	j_1	x_1, y_1, z_1	a_1, b_1, c_1, d_1
s_2	j_2	x_2, y_2, z_2	a_2, b_2, c_2, d_2
s_3	j_3	x_3, y_3, z_3	a_3, b_3, c_3, d_3
s_4	j_4	x_4, y_4, z_4	a_4, b_4, c_4, d_4
s_5	j_5	x_5, y_5, z_5	a_5, b_5, c_5, d_5
s_6	j_6	x_6, y_6, z_6	a_6, b_6, c_6, d_6

S.1:6 contain 8-character records, **J.1:6** contains 1-item integer records, **XYZ.1:6** contains 3-item double-precision records, and **ABCD.1:6** contains 4-item double-precision records. Each record of **ABCD** may be viewed as a 2×2 matrix:

$$\begin{bmatrix} a_i & c_i \\ b_i & d_i \end{bmatrix}$$

should a matrix interpretation be requested.

EXAMPLE 9.4

Read j_2 into integer variable JVAL.

```
CALL GMGETN ('R', 7, 55, 'J.2 ', 'I', JVAL, N, 0, 0, 0, 1200)
```

On successful exit: JVAL $= j_2$, N = 1.

EXAMPLE 9.5

Read x_3, y_3, z_3 into double-precision array XYZC:

```
DOUBLE PRECISION  XYZC(3)
    :
CALL GMGETN ('R', 7, 55, 'XYZ.3 ', 'D', XYZC, N, 0, 0, 0, 1200)
```

On successful exit: XYZC $= (x_3, y_3, z_3)$, N = 3.

EXAMPLE 9.6

Read only x_3, y_3 into XY(1:2).

```
DOUBLE PRECISION  XY(2)
    :
N = 2
CALL GMGETN ('R/L', 7, 55, 'XYZ.3 ', 'D', XY, N, 0, 0, 0, 1200)
```

On successful exit: XY $= (x_3, y_3)$, N = 2.

EXAMPLE 9.7

Read **XYZ.1:6** into single-precision array **XYZ** dimensioned 3×6 in the calling program.

```
REAL XYZ(3,6)
     :
CALL GMGETN ('R', 7, 55, 'XYZ.1:6 ', 'S', XYZ, N, 0, 0, 0, 1200)
```

Note that **IGAP = 0** because retrieved records are compactly stored in **XYZ**. On successful exit, **XYZ(1:3,i)** $= (x_i, y_i, z_i), i = 1, ...6$; **N = 18**. **GMGETN** automatically converts double-precision to single-precision.

EXAMPLE 9.8

Read **XYZ.1:4** into first three rows of single-precision array **CTAB** dimensioned 8×6 in the calling program.

```
REAL CTAB(8,6)
     :
CALL GMGETN ('R', 7, 55, 'XYZ.1:4 ', 'S', CTAB, N, 0, 5, 0, 1200)
```

The **IGAP** argument is here $8 - 3 = 5$. On successful exit: **CTAB(1:3,i)** $= (x_i, ...d_i), i = 1, ...4$; **N = 12**. **GMGETN** automatically converts double-precision to single-precision.

EXAMPLE 9.9

Read **S.1:4** into a character array **CS** dimensioned **CS(4)*24**.

```
CHARACTER*24   CS(4)
     :
CALL GMGETC ('R', 7, 55, 'S.1:4 ', 'A', CS, N, 0, 16, 0, 1200)
```

IGAP is $24 - 8 = 16$. On successful exit: **CS(i)(1:8)** $= s_i, i = 1, 2, 3, 4$; **N = 32**.

EXAMPLE 9.10

Read **XYZ.1:6** and **ABCD.1:6** into first seven rows of single-precision array **CTAB** dimensioned 8×6 in the calling program.

```
REAL CTAB(8,6)
     :
CALL GMGETN ('R', 7, 55, 'XYZ&ABCD.1:6 ', 'S', CTAB, N, 0, 1, 0, 1200)
```

The **IGAP** argument is now $8 - 3 - 4 = 1$. On successful exit: **CTAB(1:7,i)** $= (x_i, y_i ... c_i, d_i), i = 1, ...6$; **N** $= 42$. **GMGETN** automatically converts double-precision to single-precision.

This example illustrates key-concatenation retrieval. The general rules are:

(a) *keys run faster than cycles*, and

(b) *no gaps between records with same cycle.*

Observe that had the key specification been

ABCD&XYZ

items would have been retrieved in the order $a_i, b_i, c_i, d_i, x_i, y_i, z_i$.

EXAMPLE 9.11

(Advanced.) Read the y_i item only of XYZ.1:6 into the second row of double-precision array XYZ dimensioned 3×6.

```
DOUBLE PRECISION  XYZ(3,6)
    :
N = -1
CALL GMGETN ('R/L', 7, 55, 'XYZ.1:6 ', 'D', XYZ(2,1), N, 0, 2, 1, 1200)
```

Here IGAP = 3-1 = 2 , IOFF = 1. On successful exit: XYZ(2,i)$= y_i, i = 1, \ldots, 6$; N = 6.

EXAMPLE 9.12

Find out all there is to know about record group key ABCD.

```
CHARACTER   RNAM*20, RTYP*1
    ...
CALL GMGECY (' ', 7, 55, 'ABCD ', NR, ILO, IHI, 1500)
IF (NR .GT. 0)     THEN
  CALL GMCORN (RNAM, 'ABCD ', ILO, IHI)
  CALL GMGETN ('F/M', 7, 55, RNAM, RTYP, 0, N, M, 0, 0, 1600)
END IF
```

(GMCORN, construct a record name string; it is described in §10.6.) On GMGETN exit: RNAM = 'ABCD.1:6 ', RTYP = 'D', N = 24, M = 2.

§9.6 LIST RECORD ACCESS TABLE: GMLIRT

GMLIRT lists the Record Access Table (RAT) of a nominal dataset.

FORTRAN Reference

Calling sequence

```
CALL GMLIRT (OP, LDI, IDSN, RKEY, LISTYP, TRACE)
```

Input Arguments

OP	Option letter string. Presently D to list deleted records.
LDI	Logical Device Index of library file.
IDSN	Dataset sequence number. A zero requests the active dataset.
RKEY	If nonblank, a generally-masked record key that limits RAT display to packets with that key.
LISTYP	A one-letter string that specifies the list format. F Full listing ' ' Simplified listing
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

§9.7 PRINT RECORD CONTENTS: GMSHOR

GMSHOR prints the contents of a record, record group or table according to user-program formatting specifications. This routine replaces GMPRIN.

FORTRAN Reference

Calling sequence

```
CALL GMSHOR (OP, LDI, IDSN, RNAME, PFORM, M, IOFF, TRACE)
```

Input Arguments

OP	Print options string.
	D Place dataset name in label.
	K Print by key (all cycles).
	R Place record id in label.
	X Suppress title line.
	V TTY print with RV if on a VT-100 display terminal
LDI	Logical Device Index of library file.
IDSN	Sequence number of owner's dataset.
RNAME	Record name, possibly with masking specifications.
PFORM	Print format specification.
M	If M>0, print at most M items per record.
IOFF	Offset to first printed item; presently ignored.
TRACE	Error traceback argument.

§9.8 RENAME RECORD(S): GMRERT

Entry point **GMRERT** lets you rename record keys. Record cycles remain unchanged.

FORTRAN Reference

Calling sequence

```
CALL GMRERT (OP, LDI, IDSN, OLDKEY, NEWKEY, TRACE)
```

Input Arguments

OP	A character string containing operation specifications. Presently **W** to print a warning if no records matched.
LDI	Logical Device Index of library device.
IDSN	Dataset sequence number. A zero requests the active dataset.
OLDKEY	A character string containing the old record key left-justified blank-filled.
NEWKEY	A character string containing the new record key left-justified blank-filled.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

EXAMPLE 9.13

```
CALL GMRERT (' ', 12, 173, 'VELOCITY', 'SPEED ', 1700)
```

§9.9 PUT NAMED RECORD(S): GMPUT*x*

GMPUTN (put numerics) and GMPUTC (put characters) are the standard entry points for *creating* or *updating* named records. The operation may involve a single record or a record group, but not a table.

The two entry points are collectively described here as they share the same calling sequence. An operation key specifies various options. The database record(s) are described by the Logical Device Index, dataset sequence number, and a record or record group name. The source main storage array is specified by its address, length and datatype. Miscellaneous specifications include gaps between group records in the main-storage array, and a matrix dimension.

FORTRAN Reference

Calling sequence

```
CALL GMPUTx (OP, LDI, IDSN, RNAME, TYPE, A, N, M, IGAP, IOFF, TRACE)
```

The entry-point names are:

GMPUTC	Put character record(s).
GMPUTN	Put numeric record(s).

The arguments (all input) are:

OP A character string that specifies the operation to be performed. It consists of a mainkey optionally followed by one or more qualifiers. For example:

```
OP = 'WRITE/REPEAT'
```

Here WRITE is the mainkey while REPEAT is a qualifier. A list of valid mainkeys and qualifiers appears in Table 9.4. Mainkeys and qualifiers may be abbreviated to the "roots" shown there.

LDI Logical Device Index of library device.

IDSN Dataset sequence number.

RNAME A character string containing the name of the record or record group to be written to. Table specifications (concatenated keys) are not permitted. For safety, the name should be terminated by a blank character.

Record-update rule if qualifier U *is not given.* If a record or record group by this name already exists in the dataset, it is rewritten if (a) datatype agrees, and (b) length fits. Otherwise the existing record(s) are marked as deleted and new one(s) created.

Record-update rule if qualifier U *is given.* If update of existing record(s) is feasible, do it, otherwise skip.

TYPE The one-letter external datatype code (see Table 5.1) of record or record group to be stored. Code U (Unknown) and M (Mixed) are not permitted.

A If OP mainkey is WRITE, source array containing record(s) to be written. For a record group, successive records are adjacent if IGAP = 0, or separated by IGAP items if IGAP > 0.

If the REPEAT qualifier appears, only the first record needs to be given.

If OP mainkey is FILL, only A(1) matters so this can be a constant or single variable. The value of A(1) is used to initialize the record(s).

N For a single record, its length in items (according to TYPE).

For a record group, a positive N is the total number of items to write. A negative N implies that the length of individual records is the absolute value of N.

M If record(s) to be stored are to be viewed as rectangular matrices, set M to the first matrix dimension. Otherwise, set M to zero.

This value may be later retrieved using GMGETx (§9.5) if so desired, but has otherwise no bearing on the inner workings of GAL-DBM.

IGAP Only meaningful if OP mainkey is WRITE with no REPEAT qualifier and RNAME specifies a record group. In this case, IGAP is item gap in array A between successive records.

In all other situations, IGAP is a dummy integer argument.

IOFF Only meaningful if OP mainkey is WRITE with UPDATE qualifier given: begin updated-record writes IOFF items after record start.

In all other situations, IOFF is a dummy integer argument.

TRACE A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

EXAMPLE 9.14

The following examples illustrate the use of GMPUTN and GMPUTC for creating and updating the four record groups already used in §9.5:

S	J	XYZ	ABCD
s_1	j_1	x_1, y_1, z_1	a_1, b_1, c_1, d_1
s_2	j_2	x_2, y_2, z_2	a_2, b_2, c_2, d_2
s_3	j_3	x_3, y_3, z_3	a_3, b_3, c_3, d_3
s_4	j_4	x_4, y_4, z_4	a_4, b_4, c_4, d_4
s_5	j_5	x_5, y_5, z_5	a_5, b_5, c_5, d_5
s_6	j_6	x_6, y_6, z_6	a_6, b_6, c_6, d_6

Table 9.4 **OP Argument (Mainkeys and Qualifiers) for GMPUT*x***

Mainkey	*Root*	*Effect*
FILL	F	Initialize records with A(1) as fill item.
RESERVE	R	As FILL, but do not initialize.
WRITE	W	Write records from array A to library.
none		WRITE assumed

Qualifier	*Root*	*Valid for mainkey*	*Effect*
REPEAT	R	WRITE	All records of a record group are identical.
UPDATE	U	WRITE	Force update; skip if record(s) not found.
APPEND	A	WRITE	Force append by deleting all existing records that match record name.

The owner dataset is at sequence number 55 of library 7. **S.1:6** contain 8-character records, **J.1:6** contains 1-item integer records, **XYZ.1:6** contains 3-item double-precision records, and **ABCD.1:6** contains 4-item double-precision records. Each record of **ABCD** may be viewed as a 2×2 matrix:

$$\begin{bmatrix} a_i & c_i \\ b_i & d_i \end{bmatrix}$$

should a matrix interpretation be requested.

EXAMPLE 9.15

Create **J.1:6** and initialize it to zero.

```
CALL GMPUTN ('F', 7, 55, 'J.1:6 ', 'I', 0, 1, 0, 0, 0, 1200)
```

EXAMPLE 9.16

Store **JVAL** $= j_2$ into **J.2**:

```
CALL GMPUTN ('W', 7, 55, 'J.2 ', 'I', JVAL, 1, 0, 0, 0, 1200)
```

EXAMPLE 9.17

Create **XYZ.1:6** initialized to -1.0, and then store x_3, y_3, z_3, which are held in double-precision array XYZC(3).

```
DOUBLE PRECISION  XYZC(3)
    :
CALL GMPUTN ('F', 7, 55, 'XYZ.1:6 ', 'D', -1.0D0, 3, 0, 0, 0, 1200)
CALL GMPUTN ('W', 7, 55, 'XYZ.3 ', 'D', XYZC, 3, 0, 0, 0, 1200)
```

EXAMPLE 9.18

Create **ABCD.1:6**, the entries of which happen to be in array A2D(2,2,6).

```
DOUBLE PRECISION  A2D(2,2,6)
    :
CALL GMPUTN ('W', 7, 55, 'ABCD.1:6 ', 'D', A2D, 4, 2, 0, 0, 1200)
```

EXAMPLE 9.19

Create **S.1:6**, which is to consist of six identical 8-character records saying 'Nothing '.

```
CALL GMPUTC ('W/R', 7, 55, 'S.1:6 ', 'A', 'Nothing ', 8, 0, 0, 0, 1200)
```

EXAMPLE 9.20

Replace all six y_i items in **XYZ.1:6** (which is assumed to exist), by the second-row values of double-precision array **XYZ** dimensioned 8×6.

```
DOUBLE PRECISION  XYZ(8,6)
    :
CALL GMPUTN ('W/U', 7, 55, 'XYZ.1:6 ', 'D', XYZ(2,1), -1, 0, 5, 1, 1200)
```

10

Supplemental Operations

§10.1 GENERAL

This section describes "oddball" operations that do not fit neatly within the previous four Sections. Examples: building datasets and record names, setting processor-mode flags, controlling output volume, and so on.

Associated entry points are alphabetically ordered by the last four letters of the entry point name. A summary entry point list is provided in Table 10.1.

Note that none of these entry points has a TRACE argument. Consequently, they fall outside the scope of the NICE-DMS error handler (DMSERR, §14) and checks on correctness of input arguments is minimal or nonexistent.

Table 10.1 **Entry Points for Supplemental Operations**

Operation	*Entry Point*	*Arguments*	*See*
Set macroprocessor flag	GMACRO	MF	§10.2
Break up dataset name	GMBUDN GMUXDN	DSNAME, KEY1, KEY2, IC1, IC2, IC3 XNAME, MOK, MKC, KEY, ICYC, IREL, ICOL, IRC, MANY, IBAD	§10.3
Break up record name	GMBURN GMUARN	RNAME, RKEY, LCYC, HCYC OP, RNAME, NKEYS, KEYS, MASK, NCYCS, CYCS, COLONS, IRELC, RELCYC, IBAD	§10.4
Construct dataset name	GMCODN	DSNAME, KEY1, KEY2, IC1, IC2, IC3	§10.5
Construct record name	GMCORN GMCARN	RNAME, KEY, IL, IH RNAME, KEYS, NK, LHX, NCYCS	§10.6
Declare Page Buffer Pool	GMPOOL	PB, LP, NP	§10.7
Enter processor signature	GMSIGN	PRNAME	§10.8
Suppress open/close messages	GMSOCM	M	§10.9

§10.2 SET MACROPROCESSOR FLAG: GMACRO

GMACRO can be used to turn the "NICE macroprocessor" flag to a specific value. Setting this flag to a nonzero value affects the outcome of conditional open-library (§6.4) and close-library (§6.2) operations.

FORTRAN Reference

Calling sequence

```
CALL GMACRO (MF)
```

Input arguments

MF	0 : turns macroprocessor flag off.
	1 : turns macroprocessor flag on. Library files tagged for conditional close (§6.2) will not be closed.
	2 : as 1, plus *all* library-open operations (§6.4) will be conditional.

REMARK 10.1

The default value of this flag, on processor start, is zero.

REMARK 10.2

At the I/O Manager level (see ref. 2), this entry point is called DMACRO.

§10.3 BREAK UP DATASET NAME: GMBUDN/GMUXDN

GMBUDN receives a dataset name, and proceeds to divide it up into its five components: two character strings (mainkey and key extension) and three integers (cycles). This operation is the inverse operation to GMCODN (§10.5). 9GMUXDN performs a detailed breakdown of an external dataset name that may contain masking, cycle range, and relative cycle specifications. This entry point is too specialized for most applications. It is described here for completeness.

FORTRAN GMBUDN Reference

Calling sequence

```
CALL GMBUDN (DSNAME, KEY1, KEY2, IC1, IC2, IC3)
```

Input Arguments

DSNAME	A character string that contains the dataset name left-justified blank-filled. If the name is less than 40 characters long, it should be terminated by a blank.

Output Arguments

KEY1, KEY2	Character strings that receive mainkey and key extension, respectively, left-justified blank-filled. Each should be declared to be at least 16 character long in the calling program.
IC1, IC2, IC3	Integers that receive the three cycle numbers.

REMARK 10.3

Masking characters in the mainkey and key extension of DSNAME are permitted and will be correctly stored in KEY1 and KEY2; see Example 10.1 below.

REMARK 10.4

Masking, cycle-range or relative-cycle specifications on *cycle* components will not be correctly processed.

EXAMPLE 10.1

```
CHARACTER   KEY1*16, KEY2*16
INTEGER     I, J, K
  :
CALL  GMBUDN ('STAR.SHIP.3.8 ', KEY1, KEY2, I, J, K)
```

The outputs will be KEY1 = 'STAR ', KEY2 = 'SHIP ', I = 3, J = 8 and K = 0.

EXAMPLE 10.2

```
CHARACTER   KEY1*16, KEY2*16
INTEGER     I, J, K
:
CALL  GMBUDN ('PROC*.*.  ', KEY1, KEY2, I, J, K)
```

The outputs will be KEY1 = 'PROC* ', KEY2 = '* ', I = J = K = 0.

FORTRAN GMUXDN Reference

Calling sequence

```
CALL GMUXDN (XNAME, MOK, MKC, KEY, ICYC, IREL, ICOL, IRC, MANY, IBAD)
```

Input Arguments

XNAME	External dataset name.
MOK	One-character string. M if masking and cycle-range specifications are permitted, otherwise blank.
MKC	Maximum characters retained in keys: 4 if DAL, otherwise 16.

Output Arguments

KEY(1)	Mainkey left-justified blank-filled.
KEY(2)	Key extension left-justified blank-filled.
ICYC	A (3,2) integer array of unpacked cycle data: ICYC(K,1) Lower bound for K-th cycle (K=1,2,3) ICYC(K,2) Upper bound for K-th cycle (K=1,2,3)
IREL	(3,2) integer array of relative-cycle specs: IREL(K,1) Relative-cycle-spec for ICYC(K,1) 0=none, 1=L (lowest), 2=H (highest), 3=N (next). IREL(K,2) likewise, for ICYC(K,2). Nonzero flags may appear only for one K.
IRC	Relative cycle indicator. If a relative cycle specification for the K-th cycle is detected, IRC = K. Else IRC returns zero.
ICOL	3-integer array marking appearance of colons in cycle fields: ICOL(K) is nonzero if colon appears in K-th cycle spec, else 0.
MANY	1 if a masking or cycle-range specification appears, otherwise 0.
IBAD	Zero if no errors detected. Otherwise IBAD is set to index of character at which parsing stopped.

§10.4 BREAK UP RECORD NAME: GMBURN/GMUARN

GMBURN receives a record name, and proceeds to divide it into its three primitive components: key, low cycle and high cycle. GMUARN is the most general record name unpacking routine, as it handles the case of multiple keys and/or cycles connected through the ampersand operator. For the single-key, single-cycle case, GMBURN should be used as it is more efficient.

FORTRAN GMBURN Reference

Calling sequence

```
CALL GMBURN (RNAME, RKEY, LCYC, HCYC)
```

Input Argument

RNAME	Record name.

Output Arguments

RKEY	Character string to receive record key.
LCYC	Integer to receive low cycle.
HCYC	Integer to receive high cycle.

FORTRAN GMUARN Reference

Calling sequence

```
CALL GMUARN  (OP, RNAME, NKEYS, KEYS, MASK, NCYCS,
              CYCS, COLONS, IRELC, RELCYC, IBAD)
```

Input Arguments

OP	Options letter string: (1:1) A if "anded" keys permitted, (2:2) M if masking permitted, (3:3) H convert * in cycle spec to L:H.
RNAME	Record name.

Output Arguments

NKEYS	Number of keys stored in array KEYS.
KEYS	Record key (if NKEYS=1) or key array (if NKEYS>1). Keys must not exceed 12 characters.

MASK	MASK(I) is set to 1 if masking character (* or %) detected in the I-th key (I=1,...NKEYS), otherwise 0.
NCYCS	Number of cycle specs decoded into CYCS.
CYCS	A 3 by NCYCS integer array. Rows 1 and 2 get lower and upper cycle, respectively, row 3 gets the increment.
COLONS	COLONS(I) is the number of colons in cycle specifications (0, 1, or 2).
IRELC	1 if relative cycle specifications detected, otherwise 0.
RELCYC	A 2 by NCYCS array of relative cycle specifications; same marking scheme as used by GMUXDN.
IBAD	Index of illegal character if one such detected, otherwise 0.

§10.5 CONSTRUCT DATASET NAME: GMCODN

GMCODN receives five components of a dataset name: two character strings (mainkey and key extension) and three integers (cycles), and proceeds to pack them into a single character string suitable for presentation to entry points that receive a dataset name argument.

FORTRAN Reference

Calling sequence

```
CALL GMCODN (DSNAME, KEY1, KEY2, IC1, IC2, IC3)
```

Input Arguments

KEY1, KEY2	Character strings containing mainkey and key extension, respectively. If any of these is less than 16 characters long, the string must be terminated by a blank.
IC1, IC2, IC3	The three cycle numbers (integers).

The only output is:

DSNAME	A character string that receives the packed record name left-justified blank-filled. Allocated length in calling program should be at least 40 characters for safety.

REMARK 10.5

Masking characters in the key-string arguments (KEY1 and KEY2) are permitted; see example below.

REMARK 10.6

Masking, cycle-range, or relative-cycle specifications on *cycle* components cannot be specified with this routine.

EXAMPLE 10.3

```
CHARACTER   DSNAME*40
    :
CALL  GMCODN (DSNAME, 'STAR ', 'SHIP ', 3, 8, 0)
```

The output name will be DSNAME = 'STAR.SHIP.3.8', with right blank-fill.

EXAMPLE 10.4

```
CHARACTER   DSNAME*40
    :
CALL  GMCODN (DSNAME, 'PROC* ', '* ', 0, 0, 0)
```

The output name will be DSNAME = 'PROC*.*. ', with right blank-fill.

§10.6 CONSTRUCT RECORD NAME: GMCORN/GMCARN

GMCORN receives components of a record name: a key, a low cycle and a high cycle, and proceeds to pack them into a single character string suitable for presentation to GMGETx or GMPUTx entry points discussed in §9. GMCARN is a more general version of GMCORN. It receives a key array and an low/high/increment cycle array, and proceeds to pack all of this information into a record name.

FORTRAN GMCORN Reference

Calling sequence

```
CALL GMCORN (RNAME, KEY, ILO, IHI)
```

Input Arguments

KEY	Record key. KEY must not exceed 12 characters.
ILO, IHI	Low and high record cycle, respectively. If IHI does not exceed ILO, IHI is ignored, and will not appear in RNAME. If IHI = ILO = 0, both cycles are dropped from RNAME.

The only output is:

RNAME	A character string that receives the packed record name. Allocated length in calling program should not be less than 20 characters for safety.

EXAMPLE 10.5

```
CHARACTER   RNAME*20
    :
CALL  GMCORN (RNAME, 'SIG-XX ', 4, 0)
```

The output name will be RNAME = 'SIG-XX.4', with right blank-fill.

FORTRAN GMCARN Reference

Calling sequence

```
CALL GMCARN (RNAME, KEYS, NK, LHX, NCYCS)
```

Input Arguments

KEYS	Record key (if NK=1) or key array (if NK>1). KEYS must not exceed 12 characters.

NK	Number of keys supplied in array KEYS. NK = 1: ordinary record name. NK > 1: named record group. NK = 0: blank key assumed (has special uses).
LHX	A 3 by NCYCS array of cycle bounds and increment cycle specifications. See GMUARN for details, §10.4.
NCYCS	Number of cycle specifications (may be 0).

Output Arguments

RNAME	A character string that returns the packed record name left-justified blank-filled. Length should exceed 13*NK + 16*NCYCS characters for safety.

EXAMPLE 10.6

```
CHARACTER   RNAME*38, RNKEY(3)*12
INTEGER   LOHI(3)
    :
RNKEY(1) = 'SIG-XX'
RNKEY(2) = 'SIG-YY'
RNKEY(3) = 'SIG-ZZ'
LOHI(1) = 7
LOHI(2) = 22
LOHI(3) = 1
CALL  GMCARN (RNAME, RNKEY, 3, LOHI, 1)
```

The output name with right blank-fill will be

```
RNAME = 'SIG-XX&SIG-YY&SIG-ZZ.7:22'
```

§10.7 DECLARE PAGE BUFFER POOL: GMPOOL

Entry point GMPOOL declares a Page Buffer Pool (PBP) for subsequent use in paged I/O support. Refer to §2.3 and §4.6 of ref. 2 for details.

FORTRAN GMPOOL Reference

Calling sequence

```
CALL GMPOOL (PB, LP, NP)
```

where all arguments are input:

PB — An integer array dimensioned

LP*NP + 2*NP + 2 words

which will be used by the I/O manager as workspace for Page Buffer Pool (LP*NP words) and Page Buffer Table (2*NP words). Two words are used to store protection data.

LP — Page length in words. Must be an *exact multiple* of the internal PRU size (§2.2.4 of ref. 2) for optimal I/O efficiency. Best results are generally achieved when LP is 4 to 16 times the internal PRU (see Appendix D of ref. 2.)

If LP $\leq$ 0, the Page Buffer Pool (PBP) declaration is ignored, and no diagnostics are given.

NP — The number of pages in the buffer. As a very rough guide, NP should be of the order of 10 times the number of paged I/O devices that may be simultaneously active.

If NP $\leq$ 0, the PBP declaration is ignored, and no diagnostics are given.

REMARK 10.7

GMPOOL must be called before any paged I/O device is opened. A good place to put the call is at the start of the user program.

REMARK 10.8

Assuming that LP $>$ 0 and NP $>$ 0, GMPOOL performs the following actions: saves LP and NP, computes and saves the blank-common address of PB (which, however, does not have to be in blank common), clears the workspace, and stores protection keys.

REMARK 10.9

Once the PB array is specified using GMPOOL, the user program should never tamper with it. To do so would simply invite disaster.

REMARK 10.10

At the I/O Manager level (ref. 2), this entry point is called DMPOOL.

§10.8 SET PROCESSOR SIGNATURE: GMSIGN

GMSIGN specifies the *processor name* that will be "signed" into all datasets created by the user program. The presence of a unique signature is extremely important for high-level operation of the NICE system, but less important for non-network systems that use GAL-DBM.

FORTRAN Reference

Calling sequence

```
CALL GMSIGN (PRNAME)
```

where

PRNAME	Character string containing the processor name. Up to eight characters are permitted.

REMARK 10.11

For ordinary NICE processors, this entry point should be called once at the processor start. The name will be stored into all datasets then created by the processor.

REMARK 10.12

For NICE macroprocessors, it is usually preferable to let each component processor sign the dataset it creates. In such a case there may be several calls to GMSIGN.

REMARK 10.13

If GMSIGN is never called, the processor-name field in the Table of Contents is filled with question marks.

EXAMPLE 10.7

Specify SKYPUL82 as processor name:

```
CALL GMSIGN ('SKYPUL82')
```

§10.9 SUPPRESS OPEN/CLOSE MESSAGES: GMSOCM

Entry point GMSOCM may be used to suppress permanently or temporarily informative messages printed by the I/O manager when opening and closing logical devices (§§6.2, 6.4).

Calling sequence

```
CALL GMSOCM (M)
```

where M is the number of subsequent messages to be suppressed.

M	If M > 0, suppress the next M messages. For permanent suppression, make M large, *e.g.*, M = 10000.
	If M = 0, print is restored.

REMARK 10.14

At the I/O Manager level (ref. 2), this entry point is called DMSOCM.

11

Table Information Retrieval

§11.1 GENERAL DESCRIPTION

The Global Data Manager GAL-DBM provides a set of entry points that return state information maintained in its internal tables. Entry points that return an integer value are referenced as integer functions of the form LM*xxxx*, where *xxxx* is a mnemonic identifier. Entry points that return character information or integer arrays are referenced as GM*xxxx*.

Table 11.1 lists, alphabetically ordered by the four letters of their name, entry points which are described in §11.2 and following. Users should note that none of these functions check for legal input arguments: the calling program is assumed to insure that. If invoked with illegal arguments, such as an LDI out of range, the returned result will be meaningless.

REMARK 11.1

Information retrieval functions pertaining to error-handling, such as **LMERCD**, are covered in Section 14.

Table 11.1 **Entry Points for Information-Retrieval**

Operation	*Entry Point*	*Arguments*	*See*
Retrieve dataset creation date/time	GMCDAT	LDI, IDSN, IDT	§11.2
Retrieve deleted dataset count	LMDEDS	LDI	§11.3
Retrieve library format	GMFORM	LDI, FORM	§11.4
Retrieve dataset record keys	GMGERK	OP, LDI, IDSN, MKEY, RKEY, NKEY, TRACE	§11.5
Retrieve LDI information	GMLDI	LDI, FORM	§11.6
Retrieve active library devices	GMLIBS	LDILIB, K, M	§11.7
Retrieve dataset lock code	LMLOCK	LDI, IDSN	§11.8
Retrieve dataset name	GMNAME	LDI, IDSN, DSNAM	§11.9
Retrieve library name	GMLNAM	LDI, EDN	§11.10
Retrieve number of datasets	LMNODS	LDI	§11.11
Retrieve number of records	LMNORD	LDI, IDSN	§11.12
Retrieve number of record keys	LMNORK	LDI, IDSN	§11.13
Retrieve dataset type code	LMTYPE	LDI, IDSN	§11.14
Retrieve dataset update date/time	GMUDAT	LDI, IDSN, IDT	§11.15

§11.2 RETRIEVE DATASET CREATION DATE & TIME: GMCDAT

Entry point GMCDAT returns the creation date and time (in integer format) of a positional or nominal dataset.

FORTRAN Reference

Calling Sequence

```
CALL GMCDAT (LDI, IDSN, IDT)
```

Input Arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.

Output Arguments

IDT — where IDT is a two-word integer array:

IDT(1)	Creation date in YYMMDD
IDT(2)	Creation time in HHMMSS

If LDI does not point to a library device, or either argument is out of range, the value returned is meaningless.

§11.3 RETRIEVE DELETED DATASET COUNT: LMDEDS

Function LMDEDS returns the number of deleted datasets present in a data library device.

FORTRAN Reference

Calling Sequence

```
NDS = LMDEDS (LDI)
```

Input Arguments

LDI	Logical Device Index of library device.

Function Return

LMDEDS	Number of deleted datasets in the library (may be zero). If LDI does not point to a library device, or is out of range, the value returned is meaningless.

§11.4 RETRIEVE LIBRARY FORMAT: GMFORM

Entry point **GMFORM** returns a library format identifier given the Logical Device Index.

FORTRAN Reference

Calling Sequence

```
CALL GMFORM (LDI, FORM)
```

Input Arguments

LDI Logical Device Index of library device.

Output Arguments

FORM A character string (dimensioned at least **CHARACTER*6**). If **LDI** is not connected to a library device, or is out of range, a blank value is returned. If the input **LDI** is connected to a library, **FORM** returns one of the library format keys listed below.

DALPRO	DALPRO compatible
GAL80	Positional GAL
GAL82	Nominal GAL

§11.5 RETRIEVE RECORD KEYS: GMGERK

GMGERK scans the Record Access Table of a nominal dataset and returns a list of all record keys presently in it.

FORTRAN Reference

Calling Sequence

```
CALL GMGERK (OP, LDI, IDSN, MKEY, RKEY, NKEY, TRACE)
```

Input Arguments

OP	Operation qualifier : presently ignored.
LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.
MKEY	Maximum number of keys that can be returned.
TRACE	Error traceback argument. Do not put a zero or a negative value here; these values are reserved for internal use.

Output Arguments

RKEY	Character array containing an alphabetically sorted list of NKEY record keys.
NKEY	Number of keys returned (may be zero).

§11.6 RETRIEVE LDI INFORMATION: GMLDI

Entry point GMLDI is similar to GMFORM if the given Logical Device Index is connected to a library. If not, it returns a 4-character error key.

FORTRAN Reference

Calling Sequence

```
CALL GMLDI (LDI, FORM)
```

Input Arguments

LDI — Logical Device Index of library device.

Output Arguments

FORM — A character string (dimensioned at least CHARACTER*6). If the input LDI is connected to a library, FORM returns one of the library format keys listed below. If the LDI is not connected to a library or is out of range, a four character error key is returned.

DALPRO	DALPRO compatible
GAL80	Positional GAL
GAL82	Nominal GAL
ILDI	LDI out of range
INDI	LDI inactive
NLDI	LDI active, not connected to a library

§11.7 RETRIEVE LIBRARY DEVICES: GMLIBS

Entry point GMLIBS returns a count of active library devices and a list of their Logical Device Indices. This entry-point replaces LMLIBS.

FORTRAN Reference

Calling Sequence

```
CALL GMLIBS (LDILIB, K, M)
```

Input Arguments

M	The maximum number of libraries that can be active. Usually this is the dimension of array LDILIB in the calling program.

Output Arguments

LDILIB	A integer array of dimension M or greater. The Logical Device Indices of the active libraries are stored in the first K array locations; the remaining (M-K) entries are set to zero. If no libraries are active (K = 0), all M locations are cleared.
K	Count of active library devices (may be zero).

§11.8 RETRIEVE DATASET LOCK CODE: LMLOCK

Function LMLOCK returns returns the lock code for a dataset identified by sequence number.

FORTRAN Reference

Integer Function Reference

```
LCODE = LMLOCK (LDI, IDSN)
```

Input Arguments

LDI	Logical Device Index.
IDSN	Dataset sequence number.

Function Return

LMLOCK	Returns the dataset lock code; see Table 3.2 for details. If LDI is not a library device, or if either argument is out of range, the value returned is meaningless.

REMARK 11.2

Dataset locking is not fully implemented.

§11.9 RETRIEVE DATASET NAME: GMNAME

Entry point GMNAME returns the stored name of a dataset given the Logical Device Index of its owner library and the dataset sequence number. It is preferable to use GMGENT as documented in §7.

FORTRAN Reference

Calling Sequence

```
CALL GMNAME (LDI, IDSN, DSNAM)
```

Input Arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.

Output Arguments

DSNAM	A character string that receives the dataset name left-justified blank-fill. Passed length is assumed; consequently, the name may be truncated if the string length in the calling program is insufficient to receive the full dataset name.

§11.10 RETRIEVE LIBRARY NAME: GMLNAM

Entry point GMLNAM returns the external device name of a library file given the Logical Device Index.

FORTRAN Reference

Calling Sequence

```
CALL GMLNAM (LDI, EDN)
```

Input Arguments

LDI	Logical Device Index of library device.

Output Arguments

EDN	A character string that receives the external device name left-justified blank-fill. Passed length is assumed; consequently, the name may be truncated if the string length in the calling program is insufficient to receive the full device name.

§11.11 RETRIEVE NUMBER OF DATASETS: LMNODS

Function LMNODS returns the number of datasets present in a library device. The count *includes* deleted datasets.

FORTRAN Reference

Integer Function Reference

```
NDS = LMNODS (LDI)
```

Input Arguments

LDI	Logical Device Index of library device.

Function Return

LMNODS	Returns number of datasets in the library (may be zero). If LDI does not point to a library device, or is out of range, the value returned is meaningless.

§11.12 RETRIEVE NUMBER OF DATASET RECORDS: LMNORD

Function LMNORD returns the number of records in a positional or nominal dataset. This entry-point replaces LMRECS.

FORTRAN Reference

Integer Function Reference

```
NRECS = LMNORD (LDI, IDSN)
```

Input Arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.

Function Return

LMNORD	The number of records stored in the dataset.
	For positional datasets, this count always *excludes* the descriptor record.
	For nominal datasets, the count includes both ordinary records and record groups.
	If LDI does not point to a library device, or if either argument is out of range, the value returned is meaningless.

§11.13 RETRIEVE NUMBER OF RECORD KEYS: LMNORK

Function LMNORK returns the number of record keys in use for a given nominal dataset.

FORTRAN Reference

Integer Function Reference

```
NRK = LMNORK (LDI, IDSN)
```

Input Arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.

Function Return

LMNORK	Number of Record Access Packets in use. If LDI is not a library device, or if either argument is out of range, the value returned is meaningless.

§11.14 RETRIEVE DATASET TYPE CODE: LMTYPE

Function LMTYPE, referenced as an integer function, returns the data type code of a dataset identified by Logical Device Index and sequence number. The data type is relevant to DAL-conforming datasets only.

FORTRAN Reference

Integer Function Reference

```
ICODE = LMTYPE (LDI, IDSN)
```

Input Arguments

LDI	Logical Device Index.
IDSN	Dataset sequence number.

Function Return

LMTYPE	Dataset type code. If the LDI device is not a library, or if either argument is out of range, the value returned is meaningless.

§11.15 RETRIEVE DATASET LAST-UPDATE DATE & TIME: GMUDAT

Entry point GMUDAT returns the last-update date and time (in integer format) of a *nominal* dataset.

FORTRAN Reference

Calling Sequence

```
CALL GMUDAT (LDI, IDSN, IDT)
```

Input Arguments

LDI	Logical Device Index of library device.
IDSN	Dataset sequence number.

Output Arguments

IDT — where IDT is a two-word integer array:

IDT(1)	Last-update date in YYMMDD
IDT(2)	Last-update time in HHMMSS

If the LDI does not point to a library device, or if either argument is out of range, or if the library is positional, the value returned is meaningless.

REMARK 11.3

The date and time of last update is maintained in GAL82 libraries, but not in GAL80 or DAL libraries.

12

Copy Operations

§12.1 GENERAL

Transfer operations involve copying datasets from one library to another and records from one dataset to another. Entry points for copy operations are summarized in Table 12.1.

Table 12.1. **Entry Points for Copy Operations**

Operation	*Entry Point*	*Arguments*	*See*
Copy by name	GMCOPN	LDIS, DSNAME, LDID, IACT, 0, TRACE	§12.2
Copy nominal record	GMCOPR	OPL, LDIS, IDSNS, RNS, LDID, IDSND, RND, TRACE	§12.3
Copy by sequence	GMCOPS	LDIS, IDSN1, IDSN2, LDID, IACT, 0, TRACE	§12.4
Copy indexed record	GMCOPZ	LDIS, IDSNS, IRECS, SIZR, OFFS, LDID, IDSND, IRECD, TRACE	§12.5
Copy and rename dataset(s)	GMCORD	OPL, LDIS, DNAMS, IDSN, LDID, DNAMD, TRACE	§12.6

§12.2 COPY DATASETS BY NAME: GMCOPN

GMCOPN copies one or more datasets from one data library to another, or to the same library. Datasets to be copied are identified by name. Copied datasets are appended to those existing in the destination library. The operation may be qualified to apply to active or deleted datasets only.

FORTRAN Reference

Calling sequence

```
CALL GMCOPN (LDIS, DSNAME, LDID, IACT, 0, TRACE)
```

where all arguments are input:

LDIS	Logical Device Index of source library.
DSNAME	Name identifying dataset(s) to be copied. Often contains masking and cycle-range specifications.
LDID	Logical Device Index of destination library. LDID = LDIS is permitted.
IACT	Dataset activity qualifier: 1 Copy active datasets only. -1 Copy deleted datasets only. 0 Copy both active and deleted datasets.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 12.1

The fifth argument is presently inactive.

REMARK 12.2

Datasets are copied on a record-by-record basis.

REMARK 12.3

If deleted datasets are copied, they will be marked as deleted in the destination library.

REMARK 12.4

If the destination library contains datasets with names matching those of copied elements, the original datasets are marked as deleted.

REMARK 12.5

Copying indexed-record datasets from GAL to DAL may result in loss of information. The loss may include one or more of the following: long dataset keys, GAL-only TOC fields, descriptor record.

REMARK 12.6

Named-record datasets can only be copied from a GAL82 file to another. If the destination library is not GAL82, an error will result.

§12.1.2. GMCOPN Usage Examples

EXAMPLE 12.1

Copy all active datasets of library 3 to library 7:

```
CALL GMCOPN (3, '*.* ', 7, 1, 0, 2500)
```

EXAMPLE 12.2

Copy all active and deleted datasets of library 3 whose key extension is **FORCE** to library 7:

```
CALL GMCOPN (3, '*.FORCE.* ', 7, 0, 0, 2500)
```

§12.3 COPY NOMINAL RECORDS: GMCOPR

GMCOPR copies a nominal record or record group from a source dataset to an existing destination dataset. The destination record may be renamed. Datasets may be in the same library or on different libraries.

FORTRAN Reference

Calling sequence

```
CALL GMCOPR (OPL, LDIS, IDSNS, RNS, LDID, IDSND, RND, TRACE)
```

where all arguments are input:

OPL	Option letter string. Presently: K copy by key. W give warning message if no records found.
LDIS	Logical Device Index of source library.
IDSNS	Sequence number of source dataset.
RNS	Name of source record or record group.
LDID	Logical Device Index of destination library.
IDSND	Sequence number of destination dataset.
RND	Name of destination record or record group.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

§12.4 COPY DATASETS BY SEQUENCE: GMCOPS

GMCOPS copies one or more datasets from one data library to another, or to the same library. Datasets to be copied are identified by sequence range. Copied datasets are appended to those existing in the destination library. The operation may be qualified to apply to active or deleted datasets only.

FORTRAN Reference

Calling sequence

```
CALL GMCOPS (LDIS, IDSN1, IDSN2, LDID, IACT, 0, TRACE)
```

where all arguments are input:

LDIS	Logical Device Index of source library.
IDSN1	Sequence number of first dataset to be copied. If zero, IDSN1 = 1 is assumed.
IDSN2	Sequence number of last dataset to be copied. If zero, IDSN2 = IDSN1 is assumed. If IDSN2 exceeds the number of datasets in the source library, the copy process stops at the last dataset.
LDID	Logical Device Index of destination library. LDID = LDIS is permitted.
IACT	Dataset activity qualifier: 1 Copy active datasets only. -1 Copy deleted datasets only. 0 Copy both active and deleted datasets.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

REMARK 12.7

The sixth argument is presently inactive.

REMARK 12.8

Datasets are copied on a record-by-record basis.

REMARK 12.9

If deleted datasets are copied, they will be marked as deleted in the destination library.

REMARK 12.10

If the destination library contains datasets with names matching those of copied elements, the original datasets are marked as deleted.

REMARK 12.11

Copying indexed-record datasets from GAL to DAL may result in loss of information. The loss may include one or more of the following: long dataset keys, GAL-only TOC fields, descriptor record.

REMARK 12.12

Named-record datasets can only be copied from a GAL82 file to another. If the destination library is not GAL82, an error will result.

EXAMPLE 12.3

Copy dataset 14 of library 3 to library 7:

```
CALL GMCOPS (3, 14, 0, 7, 0, 0, 2500)
```

EXAMPLE 12.4

Copy all active datasets in sequence range 10 through 42 (inclusive) of library 4 to library 7:

```
CALL GMCOPS (3, 10, 42, 7, 1, 0, 2700)
```

§12.5 COPY INDEXED RECORDS: GMCOPZ

GMCOPZ copies an existing positional dataset record to another existing dataset. Source and destination datasets may be in different libraries. The record may be appended to the destination dataset, or overwrite an existing record. Record length and offset specifications may be given.

FORTRAN Reference

Calling sequence

```
CALL GMCOPZ (LDIS, IDSNS, IRECS, SIZR, OFFS, LDID, IDSND, IRECD, TRACE)
```

where all arguments are input:

LDIS	Logical Device Index of source library.
IDSNS	Sequence number of source dataset.
IRECS	Source record index.
SIZR	If nonzero, size of record to be transmitted. If zero, the TOC size minus offset will be used.
OFFS	Destination record offset in words.
LDID	Logical Device Index of destination library.
IDSND	Sequence number of destination dataset.
IRECD	If ≥ 0, index of destination record to be replaced. If -1, append record.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

§12.6 COPY AND RENAME DATASET(S): GMCORD

GMCORD copies from a library to another datasets that match a given name, or a single dataset at a given sequence. source and destination library may coalesce. The copied datasets may have different names. An input option permits restricting the copy operation to active datasets, deleted datasets, or both. Generally invoked with wild-card keys or characters in the name.

FORTRAN Reference

Calling sequence

```
CALL GMCORD (OPL, LDIS, DNAMS, IDSN, LDID, DNAMD, TRACE)
```

where all arguments are input:

OPL Options letter string.

- A = only active datasets will be copied
- D = only deleted datasets will be copied
- ' ' = all datasets will be copied

LDIS Logical Device Index of source library.

DNAMS If non-blank, name identifying dataset(s) to be copied. Often contains masking and cycle-range specifications.

IDSN If DNAMS is blank, sequence number of source dataset.

LDID Logical Device Index of destination library. LDID = LDIS is permitted.

DNAMD If nonblank, specifies name of destination dataset(s). Usually has masking specs.

TRACE A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

13

Text Group Operations

§13.1 GENERAL

A text group is a record group of card images. More precisely, each record of a text group is a character string.

This Section presents entry points that operate on text groups and on the old-fashioned positional text datasets. A summary entry point list is provided in Table 13.1.

Table 13.1. **Entry Points for Text Group Operations**

Operation	*Entry Point*	*Arguments*	*See*
Get Text Group	GMXGET	OP, LDI, IDSN, RKEY, LUNIT, NX, TRACE	§13.2
Put Text Group	GMXPUT	OP, LDI, IDSN, RKEY, LUNIT, RL, TRACE	§13.3

§13.2 GET TEXT GROUP: GMXGET

GMXGET copies a text dataset or text group to a card-image FORTRAN-readable, text-editable symbolic data file.

FORTRAN Reference

Calling sequence

```
CALL GMXGET (OP, LDI, IDSN, RKEY, LUNIT, NX, TRACE)
```

where all arguments are input:

OP	Options letter string. Presently: H = write heading V = use reverse video if LUNIT = 0 W = print warning if nothing found
LDI	Logical Device Index of source library.
IDSN	Sequence of dataset in which Text Group or Dataset resides.
RKEY	If library is GAL82, name of Text Group to be copied, otherwise a dummy blank argument.
LUNIT	Logical unit number of destination FORTRAN file. If LUNIT>0, must be open at time GMXGET is called. If LUNIT=0, system print file is assumed. GMXGET does not close (or endfile) this file.
NX	If 1, insert blank carriage control (1X) when on LUNIT>0. Ignore if LUNIT=0 or NX=0.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

§13.3 PUT TEXT GROUP: GMXPUT

GMXPUT copies a card-image, FORTRAN-readable data file into a data library, where it becomes a text dataset (if library is positional – *i.e.*, DAL or GAL80) or a text group (if library is nominal – *i.e.*, GAL82). The file is copied until an end-of-file (EOF) is found.

FORTRAN Reference

Calling sequence

```
CALL GMXPUT (OP, LDI, IDSN, RKEY, LUNIT, RL, TRACE)
```

Input Arguments

OP	Operation option letters. Presently, F Make fixed-length-image text dataset (type 6) Z First text group cycle will be zero if it is a one-line statement.
LDI	Logical Device Index of source library.
IDSN	Sequence number of dataset that will contain the text group. Must exist at the time GMXPUT is called.
RKEY	If library is GAL82, name of text group to be created, otherwise a dummy BLANK argument.
LUNIT	Logical unit number of source FORTRAN file. Must be open at time GMXPUT is called. GMXPUT does not close this file.
RL	Applies to Type-6 text dataset or text group only. If RL>0 make stored record length equal to RL characters (usually 80). If RL=0, prescan LUNIT image lengths and adopt minimum covering word-aligned character count. This process usually saves disk storage, but takes longer to process.
TRACE	A *positive* integer used as identifying label in error traceback prints. Do not use a zero or negative value here; these values are reserved for internal use.

14

Error Handling

§14.1 GENERAL INFORMATION

Any GAL-DBM operation invoked through an entry point that contains a TRACE argument may be aborted or only partially executed on account of error conditions detected within GAL-DBM proper, by the I/O Manager DMGASP, or by the operating system.

This section covers error processing, explains error messages, and describes entry points that NICE programmers may use to store and retrieve error-related information, and to modify default error handling. These entry points are listed in Table 14.1.

REMARK 14.1

The term *error*, used in the present context, means lack of success in performing an action. A more accurate word would be failure. However, we shall conform here to the commonly-used term, error, because that usage is universally accepted and because failure has an extreme connotation. Computer systems and baseball players commit errors.

REMARK 14.2

Much of this Section's material parallels that of §6 in ref. 2. This similarity is not accidental, as GAL-DBM and the I/O Manager DMGASP share the same error-handling facilities. The duplication is made in the interest of saving readers the trouble of going back and forth between two manuals.

Table 14.1 **Error-Handling Entry Points**

Operation	*Entry Point*	*Arguments*	*See*
Identify user subprogram	GMUSER	SUBNAM	§14.4
Test error condition	LMERCD	IERR	§14.5
Extract error information	GMEINF	OPL, IERR, EMSG, K	§14.6
Retrieve I/O status	LMIOST	J	§14.7
Defuse fatal errors	GMEASY	KERR	§14.8
Specify error terminator	GMETER	UPGERR	§14.9

§14.2 ERROR PROCESSING OVERVIEW

Error Classification

The Global Database Manager GAL-DBM finds out about error conditions in three ways:

1. A validity check within GAL-DBM fails.
2. A validity check within the I/O Manager DMGASP fails.
3. An error indication is received from the I/O component of the operating system.

Regardless of source, GAL-DBM (or DMGASP) calls the *central error management subroutine* **DMSERR**, which serves the whole of NICE-DMS. DMSERR first logs a short message on the error print file (normally unit 6). These error messages are listed and explained in §14.2.

Next, errors are classified into three types:

1. *Warning-only.* Control returns to the calling program, and execution continues. The user program may, at this point, interrogate as to the error condition, and take appropriate action.
2. *Fatal.* Program execution is terminated after more detailed printout. If the user program has specified an error-termination routine, DMSERR calls it. (Error termination routines are useful for cleanup operations such as buffer-flushing and file closing.)
3. *Catastrophic.* The run is aborted immediately. Even if an error-termination routine has been specified, it is not called.

Catastrophic errors are those that may reflect serious problems in the user program logic. For example: destruction of internal tables caused by array overspill. For obvious reasons, this error type is not controllable by the user program or affected by the run environment.

Classification of non-catastrophic errors into fatal and warning-only depends on three factors: the *run environment*, *user-program specifications*, and *operation context*. If the user program has specified nothing, the IOM uses run environment and operation context as classification criteria:

Interactive Run. A non-catastrophic error is treated as warning-only, unless a total error count maintained by DMSERR exceeds an internally set limit (usually 50). If the error count limit is exceeded, a fatal error exit is taken.

Batch Run. A non-catastrophic error is treated as fatal for most operations. Exception to this rule involves errors detected by print routines (*e.g.*, "list TOC"), as these are generally harmless.

How does NICE-DMS know about the run environment? On first entry, it queries the operating system for such information, and saves the answer in its internal tables.

The preceding "default" treatment can be modified, within certain limits, by the user program through entry points GMEASY (§14.8) and GMETER (§14.9).

Error Terminology

Applications programmers making use of NICE-DMS should be aware of the following terminology, which is used in subsequent sections.

Error code	An integer value which is set to a nonzero value when an error condition occurs.
Error key	A four-letter character string that uniquely specifies the error type.
Error message	The diagnostic text placed by DMSERR on the error print file.
Error trace stack	The ETS is a data structure optionally maintained by NICE-DMS, and which records the tree of internal calls. (The presence or absence of ETS depends on parametrization of the GAL-DBM and DMGASP Master-Source-Code preprocessing prior to compilation.)
I/O status	An integer value, or set of integer values, returned by the operating system to identify errors detected in an I/O transaction. The I/O Manager saves this value (or values) in an internal array.

§14.3 ERROR DIAGNOSTICS

Error Message Format

Error messages issued by DMSERR are of the form

***DM* Subnam: EKEY, diagnostic text**

where **Subnam** is the name of the subroutine that calls DMSERR (often the same subroutine that detected the error), EKEY is a four-letter error key, and "diagnostic text" is a short explanatory message. This message may be followed by one or two additional lines that furnish additional details such as the I/O status value.

Note the disappearance of *error code numbers* from the message. In the present NICE-DMS, error codes have less importance than in previous versions.

List of Error Messages

All possible DMSERR error messages are listed below in key-alphabetical order. Those labeled as "IOM level errors" are native to DMGASP and included here for the reader's convenience.

In the following messages, items in *italics* denote variable names or numbers that are printed as part of the error message.

CFDS, Cannot find dataset

An operation was specified on an individual dataset, which was not found in the library. This error cannot happen with operations such as "Find dataset" or "Match dataset", which explicitly return an absence indicator. For most situations, this error is viewed as a warning-only error.

CRTB, Character record too big

A character record to be read or written through GMOVEC, GMPORC or GMPOWC, exceeds the size of GAL-DBM buffers that take care of byte addressing.

DCLE, Device close error, file: *File name*

IOM level error. The operating system has reported an error during a device-close operation. This error is a very unusual condition. Track the I/O status code for further details.

DCOE, Device connect error, file: *File name*

IOM level error. This error can only occur for VAX/VMS Block I/O devices and is very unusual. The RMS level has reported an error condition when trying to carry out a file-connect service. Track the I/O status code into the RMS Manual for further insight.

DEXE, Device extend error, file: *File name*

Not presently active; reserved for future implementations.

DINE, Device inquire error, file: *File name*

IOM level error. A device-existence query performed through a FORTRAN INQUIRE statement caused an error return. Not a common one.

DIRO, Device is read-only

IOM level error. A write-record operation was attempted on a device opened in read-only mode. The operation is ignored.

DNCL, Device not connected to library

The Logical Device Index (LDI) specified in the call to an GAL-DBM entry point is active, but is not connected to a library device.

DNDA, Device is not direct access

The device type index (TYPEX) in a GMOPEN call is negative. Negative TYPEX values are used to request sequential-access devices and are therefore illegal for holding data libraries.

DNWA, Device is not word addressable

The external PRU size parameter (XPRU) in a GMOPEN call to open a GAL device was greater than 1. GAL devices must be word-addressable, so the only valid XPRU are 1, 0 or -1 (read §5.3.1 for details).

DOPE, Device open error, file: *File name*

IOM level error. A device-open operation failed. This is a common error, especially in interactive work. If declaring an existing (OLD) file, the most likely causes are:

1. Illegal file name.
2. File does not exist.
3. File is write-locked by the user program, or another program.
4. Access permission denied by file owner.

If file is created by the open operation (NEW or SCRATCH):

1. Illegal file name.
2. On some operating systems such as CDC's NOS: file name duplicates that of an existing catalogued file.
3. On VAX: file creation was attempted on a directory that denies write permission.

If the error cause is not evident, look up the status code printed on the next line in the appropriate system manual.

DOVF, Device overflow

IOM level error. An write-record operation would have exceeded the device capacity limit. The operation is aborted.

DQEX, Disk quota exceeded

IOM level error. A write-record operation is aborted by the operating system as the disk quota would be exceeded (VAX).

DSCX, Dataset capacity exceeded

Installation of a new dataset would overflow the TOC capacity (approximately 1800 datasets). As this limit is more than enough for practical applications, this error is likely to be caused by an infinite dataset-creation loop in the user program. Thus, it is considered a catastrophic error.

DSNT, Dataset is not text

A get-text operation using GMGETX names a dataset which is in the library, but is not a text dataset.

DTNA, Device type not available

A device type is not available on the IOM version being used.

DTNC, Device type not creation's

A library device resident on a disk file was created with one TYPEX and reopened with another. For example: creation TYPEX = 3 (FORTRAN I/O) and reopened with TYPEX = 0 (Block I/O). This error will be reported only if GAL-DBM was able to read the library header record (in which the creation's TYPEX is stored) despite the TYPEX mixup. For certain combinations, however, an error will occur at the operating system or FORTRAN Run-Time Library level; in such a case the I/O Manager will report it as **DOPE, Device open error**. For example, on the VAX or Univac a FORTRAN file can be read with Block I/O, but the converse is not true.

FACD, File already connected to other LDI

A device opened with OLD status is already active on another LDI. The device-open is aborted. Declaring on the *same* LDI is permitted, however, as the previous device is automatically closed. Declaring NEW or SCRATCH is also permitted on computers like the VAX, as the system simply increments the file cycle or version number.

FNGD, File not GAL or DAL

GMOPEN has been directed to open an existing file, but the file is not a library. This is noted from a header prescan. The file is closed, and the LDI remains inactive.

ILDP, Illegal device position

IOM level error. The result of a positioning operation via DMPOST or DMPAST would result in the new device position being either negative or over the device capacity limit. The new position is not stored.

ILDI, Illegal LDI

I/O Manager or GAL-DBM level error. A Logical Device Index (LDI) is outside the legal range 1 through 16. A very common error in interactive work.

ILDS, Illegal dataset name *Dsname*

A dataset name does not comply with the rules stated in §2.

ILOI, Illegal OPTX index

IOM level error. The device-assignment options index (OPTX) supplied to either DMOPEN or DMDAST is outside the legal range -6 to +14. The device-open operation is aborted.

ILOP, Illegal operation

An illegal operation key was supplied in the calling sequence of *Subnam*.

ILRS, Illegal record size

IOM level error. The size of a record presented to a record-transfer entry point is zero or negative.

ILSN, Illegal sequence number

The sequence number supplied to GAL-DBM is out of bounds.

ILTI, Illegal type index

IOM level error. The device type index (TYPEX) presented to DMOPEN or DMDAST is outside the legal range -4 to +5.

ILXP, Illegal external PRU

IOM level error. An external PRU size presented to DMOPEN or DMDAST does not exactly divide the internal PRU size (*e.g.*, internal PRU 128 words, external PRU 24 words). This can never happen if the word-addressable default is used, which is the recommended setting.

INDI, Inactive LDI

IOM level error. An I/O operation is attempted on a device that has not been previously opened.

LDTD, Logical Device Table destroyed

IOM level error. A protection key stored in front of the auxiliary storage tables has been destroyed. This is considered a catastrophic error.

LDTF, Logical Device Table full

IOM level error. Open-device request refused because all 16 slots in the Logical Device Table are in use.

MIRE, Miscellaneous read error
MIWE, Miscellaneous write error

IOM level errors. These are "catch-all" errors for data-transfer situations that cannot be easily categorized. Typically the following happens. The I/O Manager instructs the operating system or FORTRAN Runtime Library to move a particular record, and back comes the reply: that it cannot be done. There may be many reasons behind the refusal, ranging from hardware malfunction to poor software. If the cause is not immediately apparent, and usually is not, the recommended path is to write down the I/O status code printed on the next line on a piece of paper, and proceed to consult the appropriate system manual.

MROL, Modification of read-only library ignored

An operation that would have modified the contents of a library attached in read-only mode has been attempted and caught at the GAL-DBM level. For example, marking a dataset as deleted. The operation is ignored.

NRFD, No room for descriptor

An attempt was made to install or modify a descriptor record in an indexed-record dataset, but there is no space to do so. The operation is skipped.

ODDS, Operation on deleted dataset

An operation was specified on a sequence number that corresponds to a deleted dataset. This is an error only when a specific sequence number is prescribed, for example: find record 5 of dataset 124. It does not apply to *sequence range* operations (*e.g.*, list datasets 30 to 55), in which deleted datasets are automatically ignored.

PBPD, Page Buffer Pool destroyed

IOM level error. A protection key stored in front of the Page Buffer Pool has been altered. This is considered a catastrophic error.

RBEI, Read beyond end of information

IOM level error. A read operation through DMREAD or DMRAST specifies a record that extends beyond the end of information (NEXT). The operation is ignored.

RBTS, Record buffer too small

The size of the GAL-DBM utility buffer is insufficient to do certain high-level operations.

RODS, Read outside dataset *Dsname*

A read-record operation would fall partially or completely outside the boundaries of the active dataset. Faulty positioning or incorrect record size is usually to blame. The read operation is skipped. A very common error.

SONA, Sequential operation not available

IOM level error. An operation other than open or close has been specified on a sequential-access device, *i.e.*, one opened with a negative TYPEX.

TMOL, Too many open libraries

The number of simultaneously active library devices exceeds an internal parameter (normally 8). The open request is ignored.

WODS, Write outside dataset *Dsname*

Same as **RODS**, but now it applies to a write-record operation. Commonly caused by trying to append data to a "closed" dataset.

§14.4 IDENTIFY USER SUBPROGRAM: GMUSER

The first executable statement of any user-program subroutine that calls a GAL-DBM entry point should be a call to GMUSER.

FORTRAN Reference

Calling sequence

```
CALL GMUSER (SUBNAM)
```

where

SUBNAM A character string of up to eight characters that identifies the user-program subroutine (normally the subroutine name).

REMARK 14.3

This name will appear at the "base" of ETS (Error Trace Stack) printouts.

REMARK 14.4

At the I/O Manager level, this entry point is known as DMUSER (§6.3 of ref. 2), which has identical effect and the same calling sequence.

REMARK 14.5

Before any call to **GMUSER** (or **DMUSER**) is made, NICE-DMS assumes **USRPRG** as ETS-base identifier.

EXAMPLE 14.1

```
SUBROUTINE OPENDL (LDI, FILNAM, ...  )
   :
CALL GMUSER ('OPENDL')
   :
CALL GMOPEN (LDI, FILNAM, ...  )
   :
RETURN
END
```

§14.5 TEST ERROR CONDITION: LMERCD

Entry point LMERCD, referenced as an integer function, furnishes the means of testing for error conditions after a error-sensitive reference to the I/O Manager.

FORTRAN Reference

Function reference

```
IERR = LMERCD (IERR)
```

If an error condition has been detected in the previous IOM operation, a *nonzero value* is returned as both argument and function value. The double setting facilitates the use of LMERCD in conditional branching statements such as

```
IF (LMERCD(KODE) .NE. 0) CALL ERROR (KODE)
```

REMARK 14.6

There is no longer any significant correlation between the error code and a specific error type. On the contrary, the relation will frequently vary as new error conditions are introduced in NICE-DMS, because these are internally sorted (by an *ad-hoc* table-building program) alphabetically on the error key. The error code serves only two purposes: indicates the presence of error by a nonzero value; and works as a "hook" for retrieving error keys and messages through GMEINF (§14.6).

§14.6 EXTRACT ERROR INFORMATION: GMEINF

Entry point GMEINF is used to extract the error key and error message, given the error code.

FORTRAN Reference

Calling sequence

```
CALL GMEINF (OPL, IERR, EMSG, K)
```

where the input arguments are:

OPL	Operation letter. Currently M to return error message in EMSG.
IERR	Error code returned by LMERCD.

and the outputs are:

EMSG	A character string that receives the error key in its first 4 locations, followed by a comma and a diagnostic message. The total length of the text string is returned in K. If IERR is zero or is not a proper error code, ESMG is blanked and K set to zero.
K	The length of the message returned in EMSG. If the passed length of EMSG is insufficient to hold the whole message, it is truncated to that value, and K set to LEN(EMSG).

REMARK 14.7

In most cases the user program will be interested only in retrieving and testing the *error key*. The following illustrates a typical construction that tests for a device-open error.

```
CHARACTER*4 KEY
    :
CALL GMOPEN (LDI, EDNAME, DDPARS, LBTYP, 2000)
IF (LMERCD(IERR) .NE. 0)    THEN
    CALL GMEINF ('M',IERR, KEY, K)
    IF (KEY .EQ. 'DOPE')     THEN
    :
  END IF
END IF
```

§14.7 RETRIEVE I/O STATUS CODE: LMIOST

Entry point LMIOST, referenced as an integer function, returns a I/O status code in effect since the last I/O operation.

FORTRAN Reference

Function reference

```
ICODE = LMIOST (J)
```

where

J	Index to the I/O status array maintained by the I/O manager. Normally J = 1.
LMIOST	J-th entry of the I/O status array.

REMARK 14.8

These values are not only machine-dependent, but depend on whether FORTRAN I/O or Block I/O was used. In the case of FORTRAN I/O, the FORTRAN 77 standard (see ref. 7) describes a few details about I/O status codes.

§14.8 DEFUSE FATAL ERRORS: GMEASY

Entry point GMEASY (named after "take it easy") may be used to specify that the following fatal errors are to be treated as warning-only.

FORTRAN Reference

Calling sequence

```
CALL GMEASY (KERR)
```

where

KERR — If KERR > 0, treat next KERR fatal errors as warning only.

If KERR is zero, the standard error treatment of fatal errors is enforced.

If KERR < 0, treat next |KERR| fatal errors as warning-only *and* suppress all diagnostic messages. For experienced programmers only.

REMARK 14.9

Each entry to DMSERR counts as one error for the purposes of decrementing KERR.

REMARK 14.10

This entry point is primarily useful for batch runs.

REMARK 14.11

The treatment of catastrophic error conditions is not affected.

REMARK 14.12

At the I/O manager level, this is called DMEASY.

§14.9 SPECIFY ERROR TERMINATOR: GMETER

Entry point GMETER may be called to specify an error termination routine to be called in the event of a fatal error termination.

FORTRAN Reference

Calling sequence

```
CALL GMETER (UPGERR)
```

where UPGERR is the name of the error termination routine. This name must be declared EXTERNAL in the subprogram that calls UPGERR.

In the event of a fatal error condition, DMSERR calls DMFATE, which checks whether an error-termination routine has been specified via GMETER. If so, it issues the equivalent of the calls

```
CALL UPGERR ('NICE-DMS', EKEY)
```

where EKEY is the error key.

REMARK 14.13

UPGERR must not execute a RETURN. It will be futile, anyway, as the next statement in DMFATE is a call to unconditionally abort the run.

REMARK 14.14

UPGERR should not call DMSERR or DMFATE.

REMARK 14.15

At the I/O manager level, this entry point is called DMETER.

15
References

1. Felippa, C. A.: Architecture of a Distributed Analysis Network for Computational Mechanics. *Computers and Structures*, vol. 13, 1981, pp. 405-413.

2. Felippa, C. A.: *The Computational Structural Mechanics Testbed Architecture: Volume V - The Input-Output Manager DMGASP*, NASA CR-178388, 1989.

3. Felippa, C. A.: Database Management in Scientific Computing, II: Data Structures and Program Architecture. *Computers and Structures*, vol. 12, 1980, pp. 131-145.

4. Felippa, Carlos A.: *The Computational Structural Mechanics Testbed Architecture: Volume I - The Language.* NASA CR-178384, 1988.

5. Felippa, Carlos A.: *The Computational Structural Mechanics Testbed Architecture: Volume II - Directives.* NASA CR-178385, 1989.

6. Felippa, Carlos A.: *The Computational Structural Mechanics Testbed Architecture: Volume III - The Interface.* NASA CR-178386, 1988.

7. Loden, W. A.: *The NIFTY Standard: Volume II - Dataset Templates. Report No. LMSC-DO59188, Lockheed Missiles and Space Company, Sunnyvale, CA, June 1982.*

8. *Programming Language FORTRAN, ANSI X3.9-1978. New York: American National Standards Institute.*

C-3

APPENDIX A
GLOSSARY

The following quick-reference list collects terms and acronyms that often appear in the present document.

Access method	The set of procedures for accessing and transferring data structures from a residence medium to another. In the literature, the term is often used in relation to stored databases.
Active dataset	The dataset specified in the last reference to a record-level operation.
Addressing	The procedure by which a storage address at which a subsequent activity is to take place is specified.
Auxiliary storage	Storage facilities of lower cost and slower access than main storage; generally connected to the central processor by data channels.
Block	A generic term that denotes a string of storage objects such as characters, words, PRUs, etc., which are considered as a storage unit for some purpose.
Block I/O	An Input-Output process that involves direct (unbuffered) transfers of blocks of data between main storage and a disk volume. Available from many operating systems through special service entry points.
Catalogued file	Sperry terminology for permanent files whose names are maintained by the system on a Master File Directory.
Closing (a device)	See *device closing.*
Core device	A word-addressable, scratch device that resides on blank-common storage.
Current device location	A storage address maintained by the I/O Manager for each active logical device, and which identifies the location at which the next read or write operation is to take place.
Data	Information recorded on a storage device.
Database	An organized collection of operational data needed for the completion of an activity. The term is usually reserved for activities at the task or project level.
Database manager (DBM)	A data management system that interfaces a database with its user environment.

Data library — A named partition of a database.

Data management system — A software module that centralizes activities pertaining to the manipulation of a class of data structures.

Data manager — The decision-making component of a data management system.

Dataset — A record, or set of records, that is a named element of a data library.

Dataset block — In a DAL-file organization, a block of machine words transmitted to and from a dataset through Block I/O requests. A DAL-dataset block may contain one or more logical records. The distinction between blocks and records is important in DAL files because of the presence of inter-record "gaps".

In GAL-file organizations, which are word-addressable, the distinction between blocks and records disappear.

Dataset descriptor — An optional character record stored in a GAL dataset, and which contains descriptive information about its contents. In an indexed dataset, the descriptor is stored ahead of the dataset proper, and is assigned record index 0. In a nominal dataset, the descriptor may be stored anywhere.

Dataset record — An array of machine words characterized by physical adjacency within a dataset, and identified by index, or (key, index) pair.

Data space — See *storage space.*

Data structure — A set of interrelated data objects viewed as a single logical entity.

Descriptor — See *dataset descriptor.*

Device — See *I/O device, logical device.*

Device closing — A process by which facilities assigned to a logical device are released (returned) to the operating system. A freed device is *inactive.* If the device was opened as a scratch device, its contents disappear.

Device declaration — See *device opening.*

Device opening — A process by which for residence of a logical device are

	requested to the operating system. An open device is said to be *active*.
Device option index (OPTX)	An index that characterizes permanency and accessibility attributes of a logical device at time of opening.
Device type index (TYPEX)	An index that describes residence and granularity attributes of a logical device at time of opening.
Direct-access storage	A type of storage that is capable of processing data at separate locations without passing over the intervening data. Also known as *random-access storage*, connoting the property that items of data can be stored or retrieved efficiently in a random order.
DMGASP	The I/O Manager implemented for NICE-DMS.
Dynamic	A qualifier applied to certain actions, such as the declaration and freeing of storage facilities, which are performed on command from a running program. (Contrast to *static*, in which such actions are performed before or after running the program.)
Extent	A contiguously-addressed storage region; also the size of any such region.
External device name	The symbolic identifier of a logical device given to the I/O manager by the user program. For disk-resident devices, this identifier contains the external file name, and often *is* simply the file name. The external device name is only used at device declaration time; from then on the device is identified by its Logical Device Index (LDI).
External file name	The identifier by which facilities for residence of a file structure are requested to the operating system.
External PRU	The Physical Record Unit (PRU) by which the user of the I/O manager addresses a direct-access logical device.
Facilities	Storage equipment available at a computer installation.
File	See *logical file*, *physical file*.
File name	The identifier(s) by which a logical file is known to the operating system.
Flushing (a data library)	The process of writing altered header/TOC buffers to a

library file to ensure conformity of contents.

Global Access Library (GAL) — The standard data-library organization managed by GAL-DBM. It is a direct-access file organization characterized by word-addressing, data descriptors, VAX-like dataset identifiers, indexed- or named-record access, and clear separation of logical and physical data description levels.

Global database — A database residing on permanent storage, and which is accessible by a network of communicating programs.

Hardware PRU — A Physical Record Unit (PRU) that corresponds directly to the mechanical and/or electronic access characteristics of the storage medium.

Indexed record — A dataset record identified by its position within the dataset.

Information — Quantifiable knowledge.

Information structure — An organized collection of information viewed as a logical entity.

I/O Device — A storage device connected to the central processor by a data channel.

I/O Manager (IOM) — The component of a multilevel data management system that is responsible for the access method.

Internal file name — The identifier by which a file structure is referenced by a running program. It is linked to the external file name (and the Logical Device Index) at time of opening.

Internal PRU — The PRU size used by the I/O manager for requesting physical-record transfers. For Block I/O devices, it coincides with the hardware PRU. For FORTRAN I/O devices, it is the Fixed Record Length declared for direct-access devices.

Library — See *data library.*

Local database — A database attached to a running program, and which disappears when the program stops.

Local file — CDC terminology for *temporary file.*

Location — An addressable component of a storage device.

Logical device	A partition of an I/O device that is managed as a logical entity for resource-allocation and administration purposes. For auxiliary storage devices, the term is equivalent to *logical file.*
Logical Device Index (LDI)	An integer that identifies a logical device entered in the Logical Device Table (LDT) of the I/O manager.
Logical Device Table (LDT)	A table of logical devices maintained by the I/O Manager.
Logical file	The description mechanism by which logical devices residing on auxiliary storage are managed by the operating system.
Logical name	DEC term for *internal file name.*
Logical record	A record structure as seen by the applications programmer.
Logical unit	The mechanism by which the applications program(mer) refers to a logical file.
Main storage	Random-access storage facilities hardwired to the central processing unit, and may be referenced by machine-code addresses.
Manager	A software element that is primarily engaged in the administration of computing resources.
Mass storage	CDC term for online, large-capacity auxiliary storage facilities allocatable for public use.
Named record	A dataset record identified by name.
Nominal dataset	A dataset that consists of named records.
Online storage	Storage under direct control of the central processing unit.
Open (a device)	See *device opening.*
Page Buffer Pool	An area of main storage set aside for the realization of Paged I/O.
Paged I/O	An implementation of buffered I/O in which data transfers between the user-program workspace and an auxiliary storage device take into account the presence of a Page Buffer Pool in main storage.

Permanent file	A file structure that survives the execution of the process that created or modified it. A permanent file exists until it is specifically deleted by its owner, or (if lapsed) by the operating system.
Permanent file name (PFN)	CDC term for *external file name* of a catalogued file.
Physical device name	DEC term for *external file name*
Physical file	An area or set of areas of main or auxiliary storage managed by reference to a common identifier.
Physical record	A record structure as presented to the operating system services.
Physical record unit (PRU)	The addressing unit "granule" for direct-access devices. Varies according to usage level: see *external PRU, hardware PRU, internal PRU, sector.*
Positional dataset	A dataset that consists of indexed records.
Positioning (a device)	The insertion of a storage address into the Logical Device Table to update the current device location.
Random-access storage	See *direct-access storage.*
Record	A set of data items characterized by physical adjacency, which constitutes the basic transaction unit in the transmission of data between main and auxiliary storage.
Record Access Table	A directory of records held within a nominal dataset.
Record Group	A set of named records of identical length and data type, resident in the same nominal dataset, and which are identified by a common key and a nonempty cycle range.
Scratch file	A file structure that disappears when the process that created it stops, or when the file is explicitly closed.
Sector	The smallest addressable unit by the operating system on a rotating direct-access storage device such as drum or disk. It may be a true equipment characteristic (in which case it coincides with a hardware PRU) or the result of simulation by the operating system.
Sequential-access device	A type of storage in which the data can be accessed only by following the order in which it was stored.

Spanned record	A record built-up as a sequence of block writes, and which can be read back as a single record with no internal gaps.
Storage	Any device that is capable of retaining information over a period of time and of delivering it on request.
Storage address	A label, name, or number that identifies the place at which data are recorded on a storage device.
Storage device	A subset of the storage facilities that is treated as an operational entity for purposes of allocating or releasing storage resources during the execution of a run or process.
Storage peripheral	A readily detachable part of the storage facilities; for example, a magnetic tape unit or a removable disk volume.
Storage space	The region allocated or attributed to a storage device or a logical partition thereof. The size of a storage resource.
Storage unit	See *storage space.*
Table of Contents (TOC)	A directory of datasets maintained in a data library. Physically, the TOC is a matrix-like arrangement of information that is split into segments for paging purposes.
Temporary file	A file structure that disappears when the job that created it terminates, or when its facilities are released. (On many systems, temporary and scratch files are indistinguishable.)
Text dataset	A dataset that stores card-image information.
Track	The portion of a mechanical storage device such a drum, disk, or tape, which is accessible to a given read/write station.
Unit	See *logical unit, storage unit.*
Volume	The storage space associated on a one-to-one basis with a separable segment of the storage facilities; *e.g.*, a magnetic tape reel, mountable cartridge or disk drive.
Word	The standard main-storage allocation unit for numeric data. Conventionally, a word holds a single-precision floating-point value. (However, this characterization is not universally agreed upon in the world of minicomputers and microcomputers.)

APPENDIX B
INDEX

NASA

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-178387		

4. Title and Subtitle
The Computational Structural Mechanics Testbed Architecture
Volume IV - The Global-Database Manager GAL-DBM

5. Report Date
January 1989

6. Performing Organization Code

7. Author(s)
Mary A. Wright, Marc E. Regelbrugge, Carlos A. Felippa

8. Performing Organization Report No.
LMSC-D878511

9. Performing Organization Name and Address
Lockheed Missiles and Space Company, Inc.
Research and Development Division
3251 Hanover street
Palo Alto, California 94304

10. Work Unit No.
505-63-01-10

11. Contract or Grant No.
NAS1-18444

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

13. Type of Report and Period Covered
Contractor Report

14. Sponsoring Agency Code

15. Supplementary Notes
Mary A. Wright and Marc E. Regelbrugge, Lockheed Missiles and Space Company, Inc., Research and Development Division, 3251 Hanover Street, Palo Alto, CA 94304.
Carlos A. Felippa, Center for Space Structures and Controls, University of Colorado, Boulder, CO 80309-0429.

Langley Technical Monitor: W. Jefferson Stroud

16. Abstract
This is the fourth of a set of five volumes which describe the software architecture for the Computational Structural Mechanics Testbed. Derived from NICE, an integrated software system developed at Lockheed Palo Alto Research Laboratory, the architecture is composed of the command language (CLAMP), the command language interpreter (CLIP), and the data manager (GAL). Volumes I, II, and III (NASA CR's 178384, 178385, and 178386, respectively) describe CLAMP and CLIP and the CLIP-processor interface. Volumes IV and V (NASA CR's 178387 and 178388, respectively) describe GAL and its low-level I/O. CLAMP, an acronym for Command Language for Applied Mechanics Processors, is designed to control the flow of execution of processors written for NICE. Volume IV describes the nominal-record data management component of the NICE software. It is intended for all users.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Structural analysis software Command language interface software Data management software	Unclassified--Unlimited Subject Category 39

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	207	A10