BNL - 22445

THE ARCHITECTURE OF THE BNL ARCHIVE AND DISSEMINATION SYSTEM

Jack Heller

December, 1976

INFORMAL REPORT

APPLIED MATHEMATICS DEPARTMENT

BROOKHAVEN NATIONAL LABORATORY

ASSOCIATED UNIVERSITIES, INC.

UPTON, N.Y. 11973

UNDER CONTRACT NO. E(30-1)-16 WITH THE

UNITED STATES ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION

DISCLAIMER

BNL - 22445

AMD - 752

THE ARCHITECTURE OF THE BNL ARCHIVE AND DISSEMINATION SYSTEM

Jack Heller

December, 1976

INFORMAL REPORT

ABSTRACT

The Brookhaven National Laboratory Archive and Dissemination System (BNLADS) has been designed to deal with the record keeping associated with archiving and disseminating sequential files through a computer network. This data base management system (DBMS) is implemented in a host language that is a subset of PL/I which is capable of running on a variety of manufacturer's equipment under their respective operating systems.

The stored sequential files that can be dealt with by the BNLADS must be in character mode (ASCII, BCD, EBCDIC). The accessing of fields must be specified by a format description which allows for forward processing of fields only. The structure of a CASE type statement allows for a data field determining a format sequence from a set of format sequences. A data definition language (DDL) has been devised to describe the accessing sequence of stored sequential files and is comprised of the key words DO UNTIL GET IF and END and the format specifications A F and I with the FORTRAN, PASCAL or PL/I allowable syntax.

A data model definition (DMD), called in the BNLADS, the physical logical description (PLD), gives the user a view of the content of each stored sequential file and contains

(i) the organization creating the file,

(ii) the logical file name,

(iii) the physical storage character set,

(iv) the visual sticker label,

(v) any comments about the data,

and (vi) the DDL details for the file.

The DDL requires all field type references to contain the field name, so that the BNLADS can access all stored sequential files by logical field name and can write stored sequential files by stating the logical field name without the necessity of referring to formats.

The BNLADS is architected in a stratified form in which the application programs (AP) are built on the accessing procedures. Below this level, the procedures become dependent upon the compiler implementation of the host language PL/I and the operating system. In this manner, BNLADS can be available on most manufacturer's hardware.

The AP's written by users specify logical names of files, which are interfaced with the operating system via the job control language; the AP's refer only to logical field names for accessing and writing stored sequential files, the interfacing with the BNLADS being accomplished via the DDL embedded in the DMD. By this architecture of the BNLADS, we are able to obtain data independence of

the allowable stored sequential files. The uniqueness of BNLADS from most other DBMS's is that one can describe previously formatted files created by the conventional higher level languages, rather than reformatting them into some predefined structure. However, it is not intended to duplicate query and report generation capabilities of other DBMS's, but to interface with any of them via the AP's.

1. Introduction

In this report we specify and discuss the Brookhaven National Laboratory Archive and Dissemination System (BNLADS), which has been designed to cope with

(i) logical accessing of sequential files,

(ii) the dissemination of subsets of sequential files of data,

(iii) the reformatting of sequential files of data,

and (iv) the record keeping associated with these archiving and disseminating activities.

In section 2 we discuss the broad specifications and desires of the BNLADS. These desires revolve around the concept that eventually this system will operate in a nonhomogeneous computer network.

In section 3 we discuss the nature of sequential files as dealt with by the BNLADS. A data definition language[4] (DDL) is specified that is capable of describing the access sequence, formatting and logical names of the fields of data. This DDL is used by the system for accessing sequential files and formatting data for output onto user specified files. The formal specification in Baccus Naur Form (BNF) is given in section 4 with a discussion of the sematics of the DDL.

The architecture of the BNLADS, i.e. the stratified organization is discussed in section 5. This organization is used with the hope that the system will be hardware independent for those machines and operating systems which support PL/I. The version used is an intersection of the PL/I which is common to all manufacturer's considered. The architecture of the BNLADS requires a physical logical description (PLD) of sequential files of data which is used by the system, via its built in compiler, to read and write files. The BNF form of the PLD is discussed in section 6.

In section 7 we discuss the data accessing language (DAL) upon which the application programs (AP) are written. These DAL procedures are written in PL/I and can be used to write AP. Three APs that have been designed are discussed in section 8. These AP are characteristic of utility programs and deal with constructing an "archive type" file, if desired, displaying the contents, selectively, of a file and reformatting a sequential file.

2. Specifications and Desires of the Brookhaven National Laboratory Archive and Dissemination System

Under the present modus operandi of computer technology there is an ever increasing accumulation of data files which are in sequential form. The logical content and making of these data files, are described and documented in reports that reside in libraries, archives and repositories; the mechanism of accessing these files resides in the programs originally constructed to process the data; the location of these machine readable files is stored someplace known to the programmers and/or project leaders. Because the relevant portions of archived historical-type data used in conjunction with data processing equipment is stored in physically different places and because the archives and programmers vary over time, much of this data becomes, in practice, unaccessible. The Brookhaven National Laboratory Archive and Dissemination System (BNLADS) addresses itself to the problem of bringing together this ever increasing divergence of logically archived data as well as allowing for the dissemination, reformatting and logically retrieving of subsets of the archived data.

The BNLADS has been designed to satisfy the following desires:

(i) the ability to locate and retrieve data fields and files using their logical names,

(ii) the ability to locate files based on the presence of logically named fields,

(iii) the ability to conform with dissemination and format standards of ANSI[2], IWGDE[11], MCN[15], ... and other as yet undefined standards,

(iv) not require the presently collected machine readable data to be reformatted,

(v) not require the recopying of user data,

and (iv) be hardware independent for third generation computers and operating systems.

Requirements i) and ii) can be met by using the concepts of Data Base Management Systems (DBMS)[4] and allowing interfacing with other systems to take advantage of their capabilities.

In order to deal with format and disseminating standards (desire iii) defined by various organizations, the BNLADS will allow for formatting specifications to be given externally at run time in its own data definition language (DDL). This DDL contains a very small set of key words and is FORTRAN, ALGOL, PL/I like in its systax and sematics[12]. The desire and need to leave the data unchanged in format (desire iv) is also met by using the DDL which can describe the accessing performed by the programs that operate on the data.

As we will see, users of the BNLADS need not recopy their data but can store the DDL in any user chosen file, which can be used by the system, thus satisfying desire v.

In order to be manufacture hardware and software independent, the system is being implemented in that intersection of the PL/I language found to run under the current available operating systems.± With the advent of computer networks, this last desire is of importance for we can envisage data archived at one node of a computer network being transferred to another node with different hardware, and yet being "logically accessible" by the BNLADS at the receiving node.

± At present (12/1/76), the parts of the system implemented have run on CDC 6600 under SCOPE, on IBM hardware under OS and VS, and on the SPECTRA under OS.

3. Desired Organization of Machine Readable Data

In this section we consider the nature of sequential files of data and the DDL which clearly stores the accessing of recorded and "logically implied" data. Since the previous descriptions and accessing of files was under other systems, devised by programmers using FORTRAN or ALGOL or PL/I or PASCAL or ... , it will either be necessary to devise a complex DDL to deal with every eventuality or restrict the type of data with the consequent simplification of the DDL. We choose the second alternative, which we believe includes many, if not most, sequential file accessing. The sequential files that the BNLADS dealt with satisfy the following two conditions:

(i) files are in character mode,

and (ii) each field's format is determined at the time of access or by a previously accessed field and its value.

In order to understand the DDL and the motivation behind its structure, let us consider various kinds of sequential files and their visualization.

Consider a fixed field formatted file of the type shown in figure 1. Each row is considered to be a record made up of two fields whose name* is given at the head of each column while the

---

* There are various terms for the name of a field: tag, attribute, field name, class, identifier, field type, ...

Average Pressure	Average Temperature
14.70	15.1
14.92	15.3
14.86	14.5
14.83	14.7
⋮	

Figure 1

A Visualization of a Fixed
Field Sequence of Records

Day	Time	Bouy #	Average Pressure	Average Temperature
1	0.0	100	14.70	15.1
"	"	105	14.92	15.3
"	"	136	14.86	14.5
"	"	125	14.83	14.7
"	1.0	100	⋮	
"	"	105		
"	"	136		
"	"	125		
	⋮			

Figure 2

A Visualization of a Fixed
Field Sequence of Records

machine recorded information is displayed in the semi-infinite strip of the figure. This view is typical of the visualizations presented in relational data set models[3]. The ordered headings are called a relation; each record is called a value of the relation.

However this view is far too simple to describe many sequential files of data. Consider the case visualized in figure 2. Here each row has a set of implied fields, their names are the headings day, time and bouy #, while the implied values are given under the headings and apply to each record, i.e. row. Upon looking at the physical storage of the information in this visualization, we would find the data

14.70 15.1
14.92 15.3
14.86 14.5
14.83 14.7
⋮

recorded on a magnetic device, e.g. a tape. We would find the other information, hopefully in reports about the collecting and use of this data, and the documentation in the programs used to access the data.

We will refer to data on the magnetic devices as recorded values. We will refer to their headings, which name the fields logically, as recording tags. We will refer to the logical data

on the left of our visualization as external. In particular the headings will be referred to as external tags and the values will be referred to as external values.

A tag value pair is a field.* We will refer to a recorded field as being made up of a recording tag and a recorded value; we will refer to an external field as being made up of an external tag and an external value.

For example 14.70 is a recorded value; Average Pressure is a recording tag and the ordered pair average pressure, 14.70 is a recorded field. Similarly, Bouy # is an external tag, 105 is an external value and the ordered pair Bouy #, 105 is an external field. The data viewed in this manner can be given a formal set theoretic description, in which ordered tuples of tuples describe the logical sequencing of the data[9]. Other descriptions of logical data with other logical connectivities have been described by set theoretic means.[1,8,9,10] However, these descriptions, though useful for theoretic discussions of data are not easily used to describe the "logical content" and accessing sequence. We therefore choose an algorithmic structured programming type of description which is more natural to programmers who originally devised the formats and accessing sequence in their program.

---

* Our tag value pair is referred to as a stored field by Date[4].

The example in figure 2 has the following access description:

```
  Day:  DØ   1,2,... *
 Time:  DØ   0.0,1.0,...,24.0*
Bouy #: DØ   100,105,136,125*
        GET  average pressure, F(10,2), # per sq in;
             average temperature, F(10,1), deg C*
        END  Bouy # *
        END  Time *
        END  Day *
```

Obviously, the key words DØ GET F and END, and the delimiters : , * ( ) ; are not enough to describe most sequential file accesses.

The BNLADS will support the following key words:

DØ UNTIL END GET IF

and the format key words

F I A .

4. Syntax and Sematics of the BNLADS DDL

The syntax of the BNLADS DDL was devised to describe accessing of sequential files which are typical of the various programs of the past and allow for documentation of the fields implied (external) or accessed (recorded). Furthermore the structure of the DDL makes all accessing descriptions appear in structured programming form.

The DO loop statement has four forms:

i) a do for various external values,

ii) a do until a delimiter,

iii) a do for a given number of records

and iv) a do until the end of file is encountered.

Associated with every DO is an END statement with the label of the DO.

The accessing of the fields is performed with a GET statement which has two forms:

i) a GET of a set of fields,

and ii) a labeled GET which triggers a case statement.

Associated with each labeled GET is an END with the label of the GET and within this labeled GET END block is a set of labeled IF END blocks for each case.

The syntax of the BNLADS DDL accessing is

⟨DDL⟩::=‖⟨get⟩|⟨do⟩|⟨labeled get⟩‖

where ‖ ‖ means any combination in any order of the statements within the set of double vertical bars.

The GET statement is defined as

⟨get⟩::= GET␣⟨recorded field⟩

min{;⟨recorded field⟩}*
0

⟨recorded field⟩::=⟨field name⟩,⟨format⟩[,⟨text⟩]

where the ⟨field name⟩ and ⟨text⟩ are arbitrary character strings not containing the reserved delimiters .. , ; * .

The ⟨format⟩ types are those used in FORTRAN, ALGOL or PL/I:

⟨format⟩::=A⟨n⟩|A⟨n⟩|A(⟨del⟩)|I⟨n⟩|

F(⟨n⟩)|F⟨n⟩.⟨n⟩|

F(⟨n⟩,⟨n⟩)|F(⟨n⟩.⟨n⟩)

where ⟨n⟩ is an unsigned decimal integer number and ⟨del⟩ is non-numeric and terminates a variable access.

The labeled GET statement is used to trigger the case statement which allows a recorded value to determine the next sequence of formats:

⟨labelled get⟩::=

⟨label⟩.. GET␣⟨recorded field⟩*

min{⟨label⟩.. IF␣⟨value⟩*
1

, ⟨DDL⟩

END␣⟨label⟩*}

END␣⟨label⟩*

where ⟨label⟩ is any character string not containing the reserved delimiters. The purpose of the ⟨labeled get⟩ statement is to allow for data determined formats. The recorded value obtained by the labeled get if equal to the ⟨value⟩ in the IF ⟨value⟩ triggers the ⟨DDL⟩ statements in the ⟨label⟩..IF END⟨label⟩ block. If none of the values of the IF blocks match a ⟨label⟩..IF* END⟨label⟩ will be performed. If the IF* block does not appear in the ⟨labeled get⟩ block and the ⟨value⟩ does not match an IF⟨value⟩ block, an error condition is raised and the job will be terminated.

The do block has the syntax:

⟨label⟩.. DO␣

min max{⟨value⟩min{,⟨value⟩}|
 0 1

⟨value⟩,⟨value⟩...[,⟨value⟩]|

GET␣⟨field name⟩,⟨format⟩[,⟨text⟩]|

UNTIL␣⟨del⟩}*

⟨DDL⟩

END␣⟨label⟩*

If the DO block does not contain one of the above possibilities, the do block is performed until the end of record condition is raised.

If the ⟨value⟩,⟨value⟩... type is used, the values must be numeric and the difference determines the increment for the next values. Termination of the do block results when the third ⟨value⟩

is exceeded, if it is present, or to the end of file if it is not present.

The ⟨label⟩.. DØ GET form of the do block requires that the GET accesses a numeric value whose integer part determines the number of times the do block is executed.

The UNTIL ⟨del⟩, where ⟨del⟩ is any character string, determines the termination of the do block upon the first access of the value ⟨del⟩.

In Appendix I we give examples of the use of the ⟨DDL⟩ to describe accessing of sequential files; in Appendix III we give examples equivalent to the Data Definition File of the extended ERDA-ANSI[11] standard.

5. The BNLADS Architecture

The BNLADS is constructed to help with the record keeping necessary in maintaining, using and disseminating subsets of large sequential files of historical data in machine readable form. A physical logical description (PLD)(*) for any sequential file dealt with via the BNLADS is either stored as the first logical record before the user's data or in a separate file. In either case the accessing of a sequential file requires a PLD which, as we will see, contains the DDL (defined in section 4) for this file (see section 6). In figure 3 we see a visualization of the two methods of archiving sequential files for use in the BNLADS.

The BNLADS system is organized in a stratified form, where each stratification is built upon lower structural layers. In figure 4 we exhibit the architecture of the BNLADS.

In order to have hardware independence, we use a subset of PL/I which is the intersection of various manufacturer's PL/I supported by their operating system and hardware. By this restriction in implementation, we obtain a hardware independence at the expense of an increase cost of implementation. However, this increased cost is relatively small if the system is used for a sufficiently long time.

---

* Our PLD is analogous to a Data Model Definition.(4)

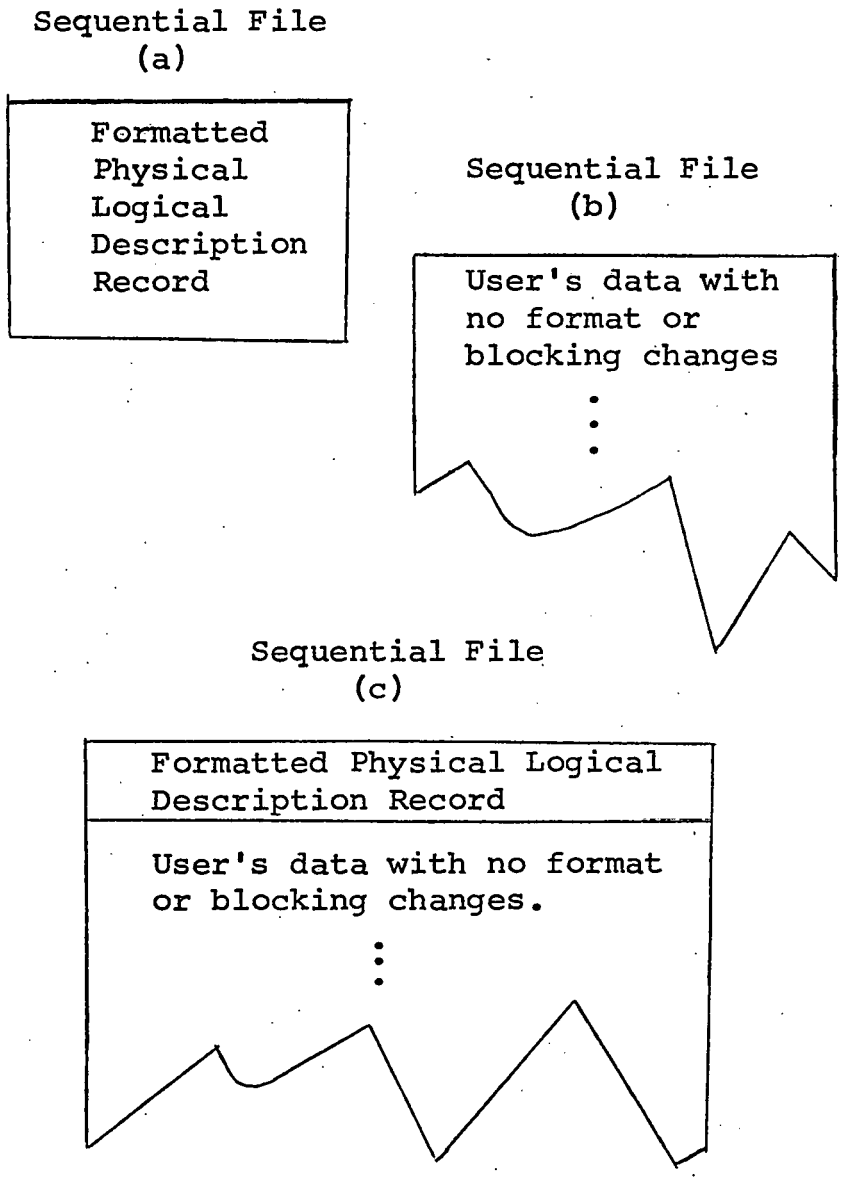

Figure 3

The two physical modes of storing sequential files. In the (a), (b) case the PLD description is physically separate from the user's data file (b). In case (c) the user's data is copied after the PLD description. The block size is determined by the user's original data file.

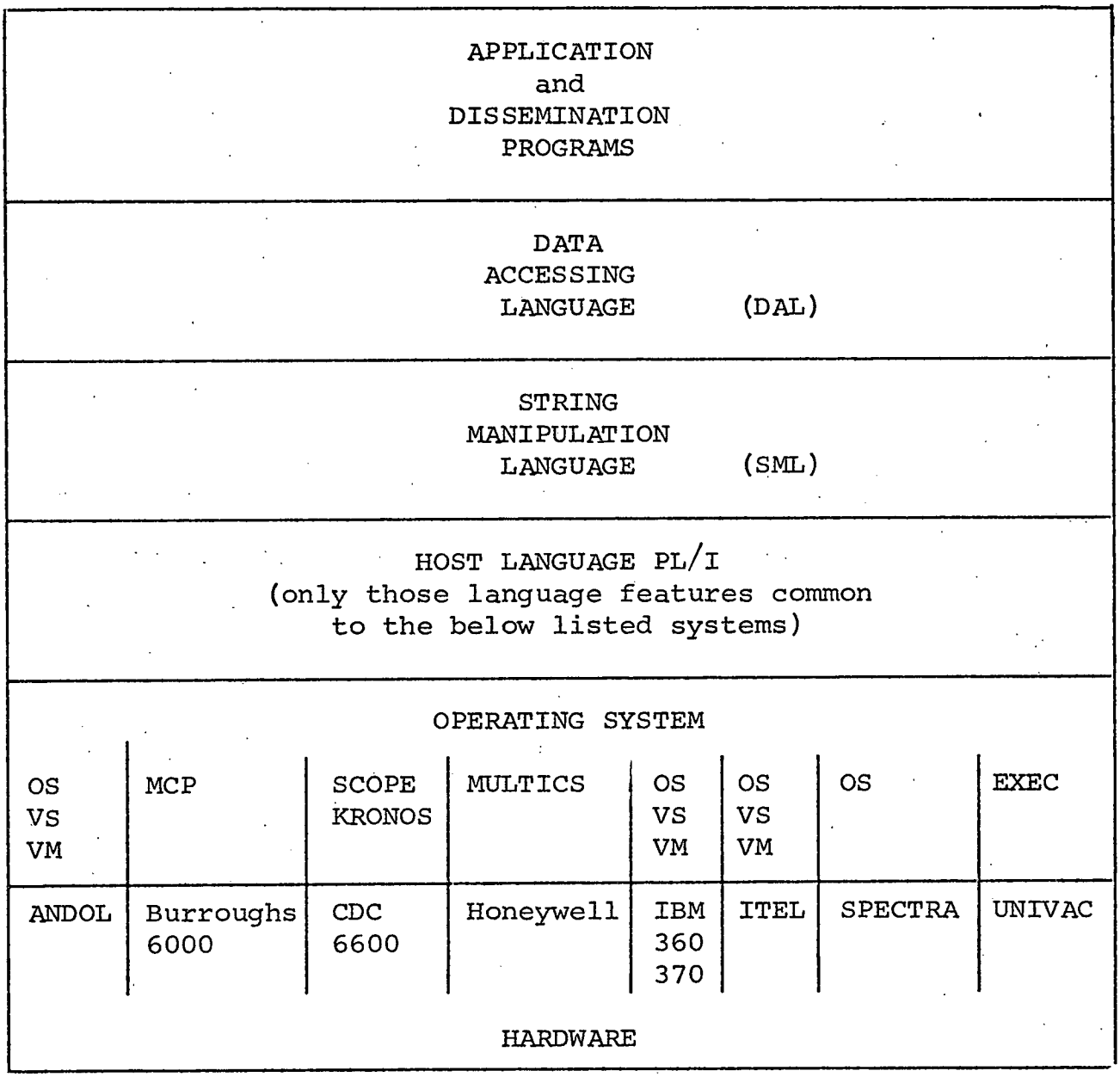

Figure 4

The Stratified Architecture of the BNLADS

Using PL/I, we develop a set of character string manipulation procedures which are found to be useful in writing the procedures and programs at the stratifications above the SML level. The SML procedures were mostly taken from a system developed on an IBM/360 system[7].

Upon the hardware, operating system, host language and string manipulation language stratifications, we build the data accessing language (DAL), a set of procedures that perform the accessing of the sequential files dealt with in the BNLADS. These procedures perform the input/output of data, formatting of output, interpretation of input format descriptions and opening and closing of files.

Finally, we build the application programs based on the lower levels of the system: DAL,SML,(*) host language, operating system and hardware.

(*) The Data Access Language (DAL) and String Manipulation Language (SML) is analogous to the Data Sub-Language (DSL) of Date.[4] The Brookhaven National Laboratory Archive and Dissemination System (BNLADS) is an example of a Data Base Management System (DBMS)[4].

6. Physical Logical Description (PLD)

In this section we discuss the PLD, its syntax and use in the BNLADS. There are two forms of the PLD: (i) the input form stated by a user and (ii) the stored form constructed by the BNLADS. Either form contains the following logical information

(i) the organization which created the file,

(ii) the title of the file,

(iii) the visual sticker label,

(iv) bibliographic references and comments,

(v) block size

(vi) character type

and (vii) the data accessing description in the DDL form.

The input description is in free form character mode in blocks of 80 characters, where one or more blanks may separate words. There is a hierarchy represented by the description (figure 5) of the PLD, which has three logical subdivisions: (i) root information which is global to the data file being described, (ii) physical information describing the particular file, and (iii) the data accessing description for this file.

Since the input description is free form, it is necessary to delimit the various logical entities. There is a default set of delimiters which can be overridden if they appear in the input description. These overrides, if present, are listed before the

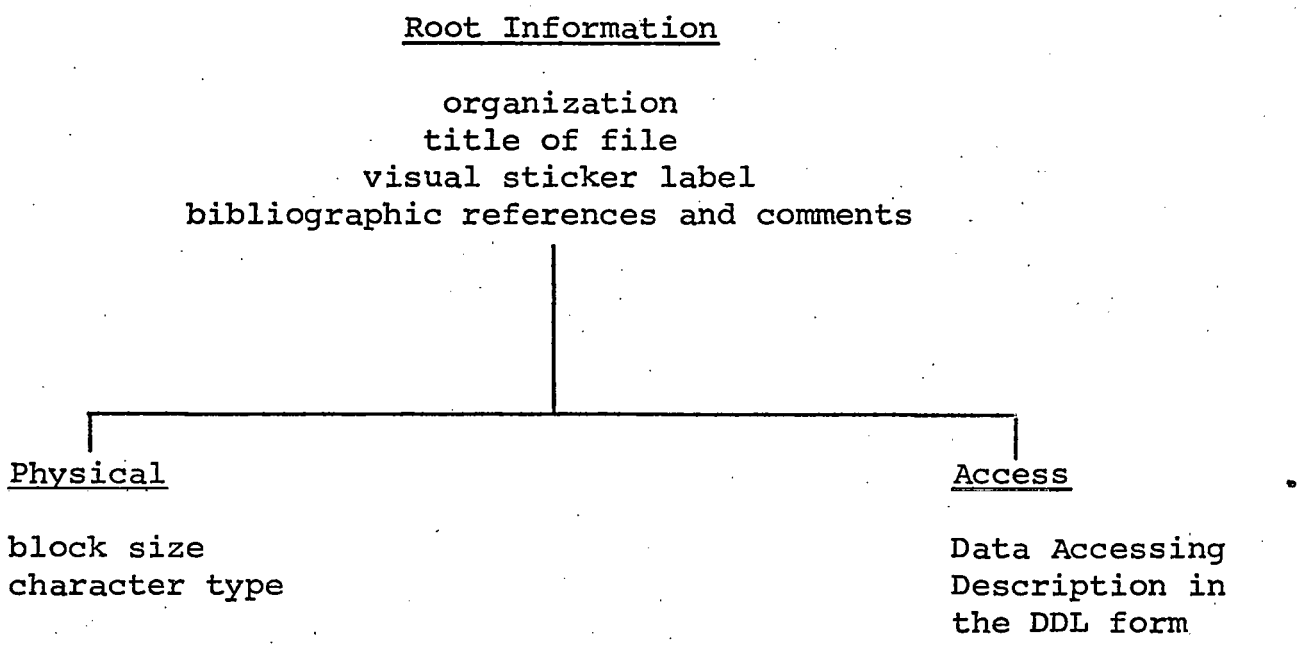

Figure 5

The Hierarchical form of the
PLD used by the BNLADS

root delimiter and indicate those delimiters that are changed with their new values.

The syntax of the PLD is

```
⟨PLD⟩::=[⟨override delimiters⟩]
    ++ROOT++b++
    ORG=⟨organization⟩{⟨vdel⟩|b}
    {PFN=|DD=}⟨file title⟩{⟨vdel⟩|b}
    VSN=⟨volume label⟩{⟨vdel⟩|b)
    min[⟨tag⟩⟨tdel⟩⟨text⟩⟨vdel⟩]
     0

    ++PHYS++ROOT++
    BLKSIZE=⟨n⟩{⟨vdel⟩|b}
    CHAR={A|B|D|E}{⟨vdel⟩|b}

    ++BNLPLD++ROOT++⟨DDL⟩⟨rdel⟩
```

The order of the fields separated by {⟨vdel⟩|b} can appear in any order within the node groups ++ROOT++b++ and ++PHYS++ROOT++.

```
[VDEL={*|⟨vdel⟩}]

[LDEL={..|⟨ldel⟩}]

[FDEL={;|⟨fdel⟩}]

[SDEL={,|⟨sdel⟩}]

[RDEL={0*|⟨rdel⟩}]   .
```

If any of the override delimiters are not present, the default values are taken and are indicated as the first character string in the choice bracket { }.

b::=any number of blanks

⟨vdel⟩::=⟨ch4⟩

⟨ldel⟩::=⟨ch4⟩

⟨fdel⟩::=⟨ch4⟩

⟨sdel⟩::=⟨ch4⟩

⟨rdel⟩::=⟨ch4⟩

⟨ch4⟩::=$\underset{1}{\min}\ \underset{4}{\max}\{$⟨character⟩$\}$

⟨character⟩::=⟨cap⟩|⟨lower⟩|⟨digit⟩|⟨special⟩

⟨cap⟩::=A|B|...|Z

⟨lower⟩::=a|b|...|z

⟨digit⟩::=0|1|...|9

⟨special⟩::=+|-|/|*|•|,|;|:|'|"|

!|#|$|%|&|(|)|=|_|

@|||⟨|⟩|...

where the ellipses ... implies any other special character readable by the particular machine being used.

⟨organization⟩::=$\underset{1}{\min}\{$⟨character⟩$\}$

⟨file title⟩::=⟨cap⟩$\underset{0}{\min}\ \underset{7}{\max}\|$⟨cap⟩|⟨digit⟩$\|$

⟨volume label⟩::=$\underset{1}{\min}\ \underset{6}{\max}\|$⟨cap⟩|⟨digit⟩$\|$

⟨tag⟩::=$\underset{1}{\min}\{$⟨character⟩$\}$

⟨text⟩::=$\underset{1}{\min}\{$⟨character⟩$\}$

⟨n⟩::=$\underset{1}{\min}\ \underset{5}{\max}\{$⟨digit⟩$\}$.

The character type of the file is indicated by CHAR= , where A implies ASCII, B implies BCD, D implies Display Code and E implies EBCDIC.

The data definition language, (DDL), which was described in section 3 is defined as

⟨DDL⟩::=‖⟨do⟩|⟨get⟩|⟨labeled get⟩‖

⟨do⟩::=⟨label⟩⟨ldel⟩DO␣⟨value⟩$_0^{\min}${,⟨value⟩}|

⟨value⟩,⟨value⟩,...[,⟨value⟩]|

GET␣⟨field⟩⟨vdel⟩|

UNTIL ␣⟨delimiter⟩⟨vdel⟩

⟨DDL⟩

END⟨label⟩⟨vdel⟩

where

⟨label⟩::=$\min_1$⟨character⟩

⟨value⟩::=$\min_1$⟨character⟩

⟨field⟩::=⟨field name⟩⟨sdel⟩

⟨format⟩

[⟨sdel⟩⟨text⟩]

⟨field name⟩::=$\min_1${⟨character⟩}

⟨format⟩::=F⟨n⟩.⟨n⟩|F(⟨n⟩,⟨n⟩)|

I⟨n⟩|F(⟨n⟩)|

A⟨n⟩|A(⟨n⟩)|

A(⟨delimiter⟩)

⟨delimiter⟩::=$\min_1$⟨character⟩

All delimiters can be surrounded by none too many blanks; all key words can be surrounded by one too many blanks. The present implementation allows any PLD to be ≤ 10000 characters in length.

The above form of the PLD is that which is used as input given by users to describe their files and used by the BNLADS to set up a list structure in the high speed storage so that accessing of the file can be performed. If the user specifies that the PLD should be stored with the user's data (figure 3(c)), an internal format is constructed, which is in the ANSI[2] magnetic tape format and is closely related to the ERDA-ANSI[11] format (see Appendix IV).

7. The Data Accessing Language (DAL)

In keeping with the design and implementation of data base management systems (DBMS), the accessing of files, formating of data, opening and closing of files and the general bookkeeping is kept away from the user. In the BNLADS we construct a set of procedures in PL/I having CHARACTER(*) VARYING arguments to perform the accessing and bookkeeping functions. Using PL/I and these DAL procedures, we then construct application programs (section 8).

We give a brief description of these procedures, which all use varying character strings as arguments. The identifier's we will use are

pld_desc	PLD character string "describing" a sequential file
pfn	title of a file
tag	a name of a field
value	character form of data in a field
XR	E for external field R for recorded field.

The procedure for input in 80 character records is invoked by CALL get80 (pfn, pld_desc, delimiter);. If the identifier delimiter is a character string of length 0, the total file whose title is pfn is read into pld_desc, otherwise the character string

up to the first occurrance of the string in delimiter is read into pld_desc. When GET80 is invoked again, it starts reading after the delimiter: this procedure simulates STREAM mode reads up to a delimiter[7].

The accessing of a sequential file in the BNLADS requires that the PLD description be given and structured in the high speed storage in a way that all other relevant procedures understand. This structuring of a character string description, is performed by

```
CALL formpld(pfn_desc,pfn);
```

The pfn_desc must contain the title of the file to be accessed (see section 6 and Appendix II), which is returned in pfn.

When a pfn is obtained from formpld, it is possible to make another copy of the structures connected with it and give this copy a new name. The procedure to perform this copy is invoked by

```
CALL copypld(pfn,new_pfn);.
```

When a pfn is obtained from formpld, it is possible to open a file for write and place the BNLADS PLD on the file as the first record. This function is accomplished by

```
CALL wrtpld(pfn).
```

If we want to start using a file without the PLD header record, but use the PLD description to write the file, we would

```
CALL wrtpldn(pfn);.
```

If a file has a PLD header record, and we want to read the file, we would

```
CALL openpld(pfn).
```

If there is no PLD header on a file which we desire to read, we would invoke the procedures formpld and then opnpldn by

```
CALL formpld(pld_desc,pfn);
CALL opnpldn(pfn);
```

After a file is opened for read by the procedures openpld or opnpldn, we can obtain the field name (tag), the field data (value) and whether the field is an external or internal field (xr) by

```
CALL gettv(pfn,tag,value,xr);
```

After a file is opened for write by the procedures wrtpld or wrtpldn, we can output the field data (value) according to the name of the field (tag) by

```
CALL puttv(pfn,tag,value);.
```

The function procedure iendfl(pfn) when invoked will return 0 if the file whose title pfn is closed but formpld(pfn) has been invoked, -1 if formpld(pfn) has not been invoked and 1 if the file whose title is pfn has been opened.

A graph of the possible sequence of DAL statements is given in figure 6. In any application program, the sequence of DAL procedures called must be a path through this graph.

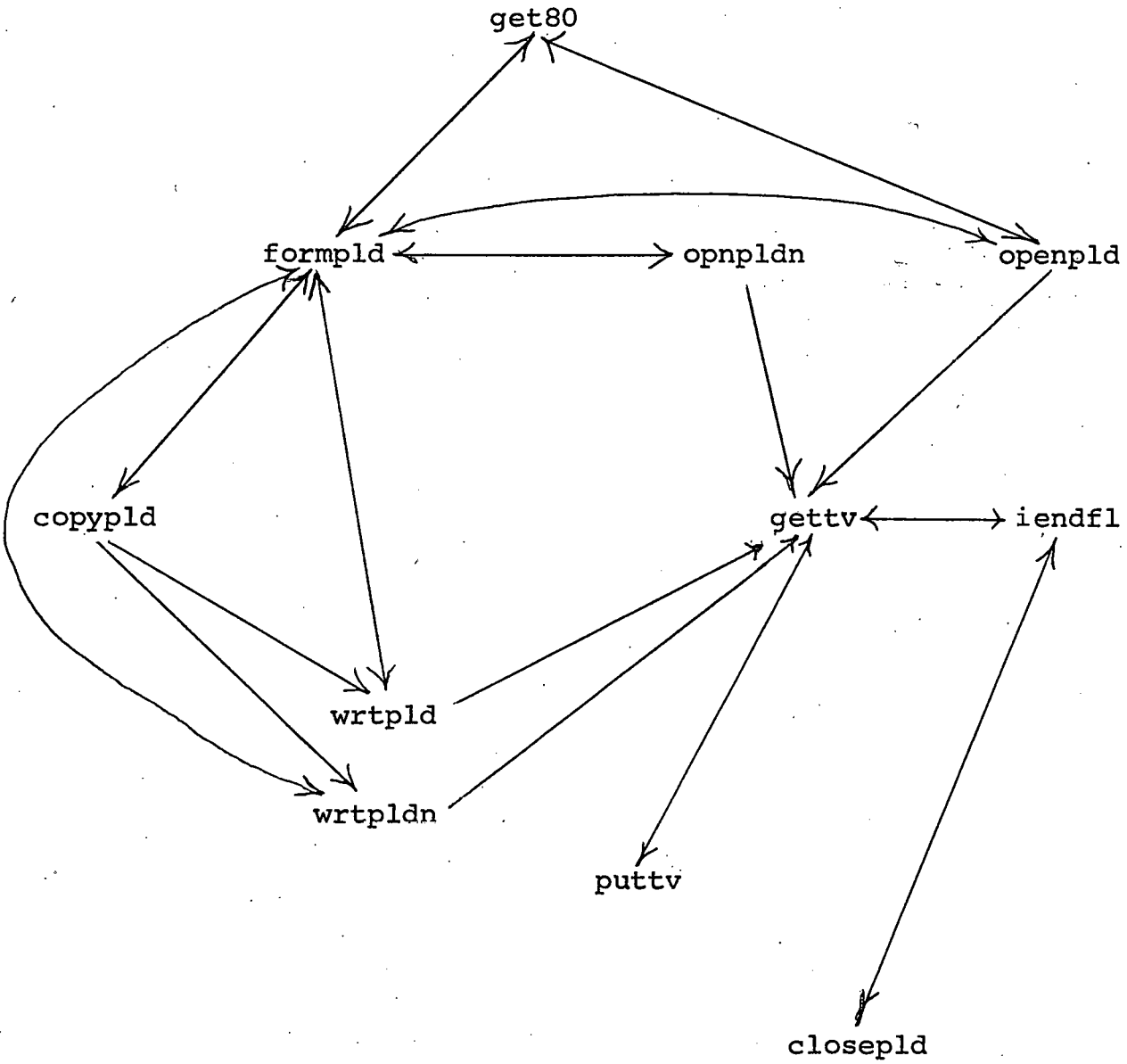

Figure 6

A graph depicting the possible sequence of DAL statements that can occur in an application program. Any application program must follow circuits in the graph and supply the semantics to deal with the data.

8. Application Programs

Any number of application programs (AP) can be written using the DAL of the BNLADS. Most AP deal with the semantics of the data retrieved from sequential files in a manner that is characteristic of a user's special view. However, some application programs (utilities) have wide use. In this section we describe a few of these utility application programs, their use and construction.

All application programs require input specifications which convey the following logical information to the AP:

i) description file titles

ii) PLD descriptions

iii) special options for the AP

and iv) special options for debugging

The standard input device is used to read the AP specifications in 80 character unordered free field records (i.e. stream mode). The general form of the input specification is

⟨AP specs⟩::=[DEL={*|⟨ch4⟩}]

$\min_{1}$ {IPFN=⟨input description title⟩⁄

⟨input data file title⟩

[⟨special AP specs⟩]{*|⟨ch4⟩}

```
min [OPFN=⟨output description title⟩
0
    ⟨output data file title⟩
    [LABEL={YES|NO}]
    [⟨special AP specs⟩]{*|⟨ch4⟩}
    [TRACE={YES|NO}[*|⟨ch4⟩]]
    [⟨key wd⟩=⟨other specifications⟩{*|⟨ch4⟩}].
```

Every application program in the BNLADS must have an input PLD and may require an output PLD description. The key word LABEL= indicates whether there is a PLD label to be placed in front of the sequential file (case (c) figure 3) and if the key word is not present YES is assumed as the default.

The key word TRACE=, if not present, has a default of NO, and indicates whether trace information should be printed on the standard output device. This key word is usually used for debugging.

The ⟨input description title⟩ and the ⟨output description title⟩ are the titles of the files which contain their respective PLD; the ⟨input data file title⟩ and the ⟨output data file title⟩ are titles of data files. If ⟨input description title⟩ and ⟨input data file title⟩ are the same, the AP assumes that the PLD is in front of the data on the sequential file (case (c) figure 3).

The order of the IPFN= and OPFN= specifications are arbitrary and each such specification is terminated by a delimiter string.

If the key word DEL= is not present, * is assumed as the delimiter, otherwise the specified

$$\langle ch4\rangle ::= \min_{1}^{} \max_{4}^{} \{\langle character\rangle\}$$

is taken as the delimiter.

Any AP, which must specify input other than those described above, will define its own key word

$$\langle key\ wd\rangle ::= \min_{1}^{} \max_{4}^{} \{\langle ch4\rangle\}=$$

and ⟨other specifications⟩ whose SYNTAX will be characteristic of its own execution.

As examples, we will describe three APs:

i) ARCHIVE,

ii) DISPLAY,

and iii) REFORM.

ARCHIVE. The ARCHIVE program takes a user's data file and user's PLD and places them on an output sequential file in the form usable by any other BNLADS AP (case (c) figure 3). There are two input files, the user's data file and the user's PLD description file; there is one output file, the BNLADS cononacol form of a sequential file (case (c) figure 3).

The standard input specifications for the ARCHIVE program is

⟨ARCHIVE specs⟩::=[DEL={*|⟨ch4⟩}]

IPFN=⟨input description title⟩b

⟨input data file title⟩{*|⟨ch4⟩}

OPFN=⟨input description title⟩b

⟨output data file title⟩{*|⟨ch4⟩}

[S2K=⟨S2K output file title⟩{*|⟨ch4⟩}].

The S2K= key word, if present, indicates that the input data file PLD is to be parsed, formatted and output onto the sequential file whose title is ⟨S2K output file title⟩. This sequential file is in the input form for introduction into a S2K data base.[5,13] This data base can be used as a user's index for locating sequential files by organization, field name and visual sticker number.

An example of ⟨ARCHIVE specs⟩ input is

IPFN=MCND ARCINP *

OPFN=MCND MCNARC1 * .

In this specification no record will be made for S2K input, the input description PLD is found in file MCND (see variable unordered fields in Appendix II), the input data is found in the file whose title is MCNINP and the resultant output is to reside on the file whose title is MCNARC1.

DISPLAY. The DISPLAY program prints the contents of a sequential file which is in the BNLADS archive form produced by the ARCHIVE program (case (c) figure 3) or uses a PLD file and a

user's data file to print the contents of the sequential file (case (a) and (b) figure 3).

The input specifications are

```
IPFN=[DEL={*|⟨ch4⟩}
      ⟨input description title⟩b
      ⟨input data file title⟩b
      [LABEL={YES|NO}]{*|⟨ch4⟩}
[AMT=[REC=⟨n⟩b]
      [CONV=⟨n⟩b]
      [BLANK=⟨n⟩]{*|⟨ch4⟩}].
```

The PLD will be printed and if the key word AMT= is present, some or all of the data file whose title is ⟨input data file title⟩ will be printed. If AMT= is not present, only the PLD will be displayed. When REC= is present, the first ⟨n⟩ records will be displayed with their field names. When CONV= is present, if more than ⟨n⟩ conversion errors are detected in the data file, the job will be terminated. When BLANK= is present, if more than ⟨n⟩ blank fields are detected the job will be terminated.

A typical ⟨DISPLAY specs⟩ is

```
IPFN=DEL===b
MCNAR1bMCNAR1==
AMT=REC=50bBLANK=10==
```

Since ⟨input description title⟩ and ⟨input data file title⟩ are the same (MCNAR1), the display program assumes that there is a BNLADS label on the file whose title is MCNAR1. The first fifty records will be displayed and if more than ten blank fields are encountered, the job will be terminated.

REFORM. The REFORM program constructs a new sequential file according to the format of the user's specified output PLD. The sequential order of the fields on the input file must be the same as the order on the output file, but not all fields need be outputted and formats need not be the same.

The input specification is

⟨REFORM specs⟩::=[DEL={*|⟨ch4⟩}b]
IPFN=⟨input description title⟩b
⟨input data file title⟩
{*|⟨ch4⟩}
OPFN=⟨output description title⟩
⟨output data file title⟩
[bLABEL={YES|NO}]
[AMT=[REC=⟨n⟩b]
[CONV=⟨n⟩b]
[BLANK=⟨n⟩]{*|⟨ch4⟩}]

If the ⟨input description title⟩ and the ⟨input data file title⟩ are the same, we assume that a PLD description heads the data. If

LABEL= is not present, we assume that the output data file is to have its PLD description in front of the data (case (c) figure 3). The key words AMT=, REC=, CONV= and BLANK= has the same meaning as in DISPLAY.

Appendix I

Examples of BNLADS DDL

We give various examples typical of accessing sequential files which have been taken from already existing applications. The simplest case is that of repeated fixed field logical records, which we start with and give successively more complex cases dealing with data dependent formats (case statement) and variable unordered fields.

Fixed Field. Let us consider a sequential file made up of fixed field records of length 24, containing the stored fields bouy #, year, day, time, pressure and temperature. Each logical record has the same format and the data on the file is terminated by an end of file mark. There is one accessing do block and its specification gives the field names, their formats and statements about the fields:

```
Long Island Sound..DO *
    GET Bouy #, A3, Ref. NC 137-A;
        year, I2;
        day, I3, Julian;
        time, I5, minutes 0 = midnight;
        pressure, F(6.2), kilos/cm sq;
        temperature, F(5.2), degs cent*
    END Long Island Sound *
```

This DDL description implies that the accessing continues until the end of the data file is encountered.

Fixed Field with External Variable. In the previous example the Bouy # was recorded in the file. If this were not the case, we might have a description where the Bouy #s are external values specified for each logical record access:

```
Long Island Sound..DO *
    Bouy #..DO  A31,A46,LØ1, LR3,RR5 *
       GET year,I2;
           day,I3,Julian;
           time,I5,minutes.0=midnight;
           pressure,F(6.2),kilos/cm sq;
           temperature,F(5.2),degs cent*
       END Bouy # *
    END Long Island Sound *
```

We note that the label of the DØ key word Bouy # is an external tag and A31 A46 AØ1 LR3 and RR5 are external values.

Variable Unordered Fields of Records. Let us consider a sequence of fields made up of a tag, i.e. the name of the field and a value, i.e. the data in that field. The tag and value data of each field is variable. The tag comes first and is terminated by a : and the data value is terminated by ==. Each record is terminated by a 0==. Data of this type is typical of the Museum Computer

Network[15] and United Nation Indexing project[14]. A typical input data stream would look as follows:

```
Artist: Miro,Jean==Title: The Night
Dancers==Owner:Irvine,Hyman==Medium:
tapestry==material:wool==size:130cmX
196cm==0==Artist:Picasso,P.==Location:
The Museum of Modern Art; N.Y.C.==
Title:Boy with Horse==Material:oil==
Material:canvas==0== ...
```

The DDL is

```
MCN Input.. DØ *
record.. DØ UNTIL 0==*
   GET MCN tag, A(:),Ref Vance D. "Manual for
     preparation of Museum Data, CCAL
     Report 3, 1976;
     data value,A(==)*
     END record*
   END MCN input*
```

Fixed Field Variable Number of Fields. Often sparse matrices are input in a format of variable number of fixed fields per record, in which the row and column position is given before the matrix element. Non-zero elements are not inputted. Suppose the matrix input were in card form image, the first card containing the number

of rows and columns of the matrix and the remaining cards containing three-tuples of row, column and matrix element value. The BNLADS DDL description would be

```
        GET rows,I5;columns,I5;
            skip,A(70)*
    matrix..DØ
        GET row,I5;
            column,I5;
            data value,F10.4*
        END matrix*
```

The field named skip is needed because the BNLADS DDL specifies STREAM mode input; therefore, after the rows and columns data are read, the remaining seventy blanks must be read, for the row, column and data value start on the next card.

If more than one matrix were in the input stream, and if the rows and columns were given and then immediately followed by the row, column data elements until XXXXX termination of a matrix, we would have the DDL description:

```
        all matrices.. DO *
            GET rows,I5;columns,I5*
        matrix.. DØ UNTIL XXXXX*
            GET row,I5;
                column,I5;
                data value,F10.4*
            END matrix *
        END all matrices *
```

Data Determined Format Choice. Many examples exist in which a field is read which then determines the format of a next set of fields. For example, consider an address file of card images. The number of lines for the address will vary, and the formats for each line will vary. In the following example, column 1 determines the type of format for the remainder of the card: 1 is name, 2 is address, 3 is city, state and zip, 4 are keys associated with the address and a 0 separates each address set of lines. The DDL statements are

```
all addresses.. DØ *
    cases..GET type,A(1)*
        name..IF 1 *
            GET last name,A(30);
                first name,A(10);
                other names,A(39)*
            END name*
        address..IF 2 *
            GET #,A(9);
                street,A(40);
                apt,A(10);
                floor,A(10);
                other,A(10);
            END address*
```

```
        location..IF 3 *
            GET city,A(39);
                state,A(30);
                zip,I10;
            END location*
        keys..IF 4 *
    each key..DO 1,2,...,79*
            GET key,A(1)*
            END each key *
          END keys*
        terminator.. IF 0 *
            GET skip,A(79)*
            END terminator*
          END cases*
        END all addresses*
```

Hierarchical Variable Fields. Many examples of data organization require a hierarchy because there is a necessity to indicate the logical connectivity of parts of a set of data representing a user define logical record. For example, suppose we are collecting bibliographic data about articles in a journal. Each issue will be viewed as a record composed of a variable number of article subparts. The record is viewed in a hierarchical fashion (see

figure 6). If the sequential file representing these variable strings were separated by an = and each variable set of authors and subjects were separated by 0= and each issue were separated by 00=, we would key board the data as follows:

```
18 12=CACM=December,1975=Programming
Languages, Natural Languages, and Mathematics=
Nour, Peter=0=Analogies related to social aspects=
language quality=language development=artificial
auxiliary languages=0=Exception Handling:
Issues and a Proposed Notation=Goodenough,J.B.=
0=mutilevel exit=GOTO statement=error conditions=
structured programming=0=The intrinsically
Exponential Complexity of the Circularity Problem
for Attribute Grammars=Jayayeri, M=
Ogden, W.F.=Rounds, W.C.=0=Attribute
grammars=circularity problem=context free
grammars=computational complexity=exponential
time=0=

                    ...

Automatic Data Structure Choice in a Language
of Very High Level=Schwartz, J.T.=0=
program optimization=automatic programming=
high-level languages=0=00=18 10=
CACM=October, 1975=a Preliminary System
for the Design of DBTG Data Structures=
Gerritsen, R=0=network model of data bases=
Data Base Task Group=data base design=data
structure=0=

                    ...

Horner's Rule for the Evaluation of a
General Closed Queueing Network=
Reiser, M=Kobayashi, H=0=Queueing
Networks=queueing theory=Horner's Rule=
service rate=0=00=

                    ...
```

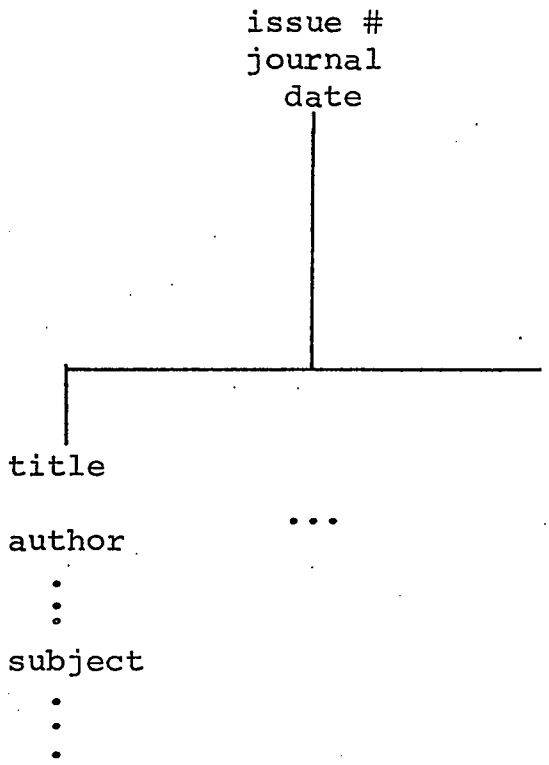

Figure 6

A hierarchical record in which each article contains one title, a variable number of authors and a variable number of subjects.

The DDL for this type of sequentially stored hierarchical type of data is

```
all issures..DØ *
      GET issue #,A(=);
          journal,A(=);
          date,A(=)*
articles..DØ UNTIL 0==*
      GET title,A(=)*
authors..DØ UNTIL 0=*
      GET author,A(=)*
      END authors*
subjects..DØ UNTIL 0=*
      GET subject,A(=)*
          END subjects*
      END articles*
      END all issures*
```

Mixed Fixed Field and Variable Unordered Field of Records. It is possible to have both fixed field and free field data in the same logical record. An example of mixed fixed and variable data fields is given by the ERDA-ANSI standard for data transmission[11]. The BNLADS DDL and its relation to the ERDA-ANSI DDF and DF is discussed in Appendix III.

Appendix II

BNLADS PLD Input Examples

The following examples have been formed from existing data files and descriptions.

Fixed Field File. The following case was taken from a form given to user's of the meteorology tapes disseminated by BNL. The names of the fields and formats are those given on the form.

```
VDEL=== RDEL=0==
++ROOT++ ++   COMMENT CLIMATOLOGY TAPES==
    PFN=DATACL ==  ORG=BNL MET == VSN=N12345 ==
CONTACT TISCHLER, JOYCE BNL MET==
COMMENT APRIL 1967 - CURRENT==
REF SINGER, I. A. AND SMITH, M. E.,
 JN. METEOR. VOL. 10, NO. 2, APR.1953,P. 121-126.==
++PHYS++ROOT++  BLKSIZE= 83== CHAR= B==
++BNLPLD++ROOT++
 INPUT.. DO ==
GET   -60 IN TEMPERATURE DEG C,F4.1;
      -30 IN TEMPERATURE DEG C,F4.1;
      -3 IN TEMPERATURE DEG C,F4.1;
      SHELTER IN DEG C, F4.1;
      37' TEMPERATURE,F4.1;
      75' TEMPERATURE,F4.1;
      150' TEMPERATURE,F4.1;
      300' TEMPERATURE,F4.1;
      410' TEMPERATURE,F4.1;
      27' DIRECTION DEGREES,F3.0;
      37' SPEED,F3.1,METERS/SEC.;
      150' DIRECTION,F3.0,DEGREES;
      150' SPEED,F3.1,METERS/SEC.;
      355' DIRECTION,F3.0,DEGREES;
      355' SPEED,F3.1,METERS/SEC.;
      GUSTINESS,I2,1=A 2=B1 3=B2 4=C 5=D;
      NET RADIOMETER LANGLEYS,F5.0;
      PYRHELIOMETER LANGLEYS,F6.0;
      PRECIPITATION,F4.0,INCHES;
      SKIP,A(1),BLANK NOT USED:
      TIME,I4,HOUR*10;
      YEAR-MONTH,A2,JAN=A FEB=B ... DEC=L -- 1960=A 1961=B ...;
      REL HUMIDITY,F5.0==
     END INPUT==
      0==
```

Variable Unorder Fields. The following description is typical of textual data that is input to the GRIPHOS data base management system used by the Museum Computer Network and the United Nations Library.[14,15]

```
REF VANCE, D.; MCN DATA PREPARATION MANUAL;
 CCAL PERT. 7 (1975)*
REF HELLER,J.; GRIPHOS PROGRAMMER'S GUIDE;
 MCN PUBL. (1974)*
CONTACT HELLER, J.*REMARK THIS DATA INPUT WAS USED
 TO CHECK OUT THE ARCHIVE PROGRAM VERSION 1.1
 AT BNL 9/76*
++ROOT++ ++ PFN=ARCINP *VSN=GRIT01*ORG=MCN*
++PHYS++ROOT++BLKSIZE=80     *CHAR=B*
++BNLPLD++ROOT++ TOTAL..DO*
RECD..DO UNTIL 0==*
 GET TAG,A(6);VALUE,A(==)*
 END RECD*
 END TOTAL*
0*
```

Appendix III

The Relation between the BNLADS PLD and the ERDA-ANSI DDF and DF

There are similarities and differences between the PLD of the BNLADS and the DDF and DF of the extended ERDA-ANSI data exchange format[11]. However, there is no <u>logical</u> inconsistency between the two concepts, the differences being syntactical, physical storage and semantical. In this appendix we give the BNF form of the DDF and then we consider some of the examples in the ERDA report[11] giving their corresponding PLD form.

The BNF specification of the DDF is

⟨ddf⟩::=[⟨control⟩@]
[⟨name⟩@]
[$\min_{1}$[⟨row label⟩%]@]
[$\min_{1}$[⟨column label⟩%]@]
[⟨format⟩]#.

The sematic meaning of the ⟨ddf⟩ is described in the report[11] section "Draft Proposal An Extendable Standard for Information Interchange on Magnetic Tape."

In this report Appendix A "Explanatory Examples" there are illustrations of DDFs and DFs, which we will state and then give the PLD form which will document and describe the accessing of the DF.

Example A.1.1.1 Unstructured Data Element

```
DDF      AUTHOR#
DF       DOE, JANE#.
```

The corresponding PLD for the above DDF could be

```
++ROOT++b++
Reference Erda Interlaboratory Working
Group for Data Exchange (IWGDE)
edited by Deane Merrill and Donald Austin
September 1976 LBL-5329*
++PHYS++ROOT++CHAR=B*
 BLKSIZE=80*
++BNLADS++ROOT++
 GETbAUTHOR,A(#)* 0*
```

In this PLD description, we have assumed that the data element Doe, Jane is the DF and we have taken the liberty to give a reference about this example. If the data file contained a sequence of authors separated by a #, we would replace the PLD access description statement

```
GETbAUTHOR,A(#)*  by
        all authors..DØ*
           GETbAUTHOR,A(#)*
           END all authors*.
```

In the remaining examples, we will assume that there is a sequence of fields comprising the DF up to a file terminator.

Example A.1.22 Vector Fields.

1) Delimited Text Vector with Name and subfield Labels

```
DDF     1000&&&&@MAILING ADDRESS@
        NAME%STREET%CITY%STATE%ZIP@#
DF    DOE, JOHN@1010 MAPLE ST.@OSHGOSH@OHIO@12345#
```

The corresponding access description within the PLD for the above DDF is

```
MAILING ADDRESS..DØ *
        GET  NAME,A(@);
             STREET,A(@);
             CITY,A(@);
             STATE,A(@);
             ZIP,A(#);
        END  MAILING ADDRESS*
```

2) Formatted Integer Vector with Name and Subfield Labels

```
DDF     1100&&&&@POPULATION@CY60%CY65%
             CY70%CY75@(416)#
DF      765432987345903231897654#
```

The corresponding BNLADS access description is

```
POPULATION..DO *
        GETB    CY60,I6;
                CY65,I6;
                CY70,I6;
                CY75,I6*
        END POPULATION*.
```

A.1.2.3 Array Fields

1) Delimited Text Array with Name and Cartesian Label

```
DDF     2000&&&&@PROPERTIES@GOLD%
        SODIUM%COPPER@DENSITY%
        COLOR%ACTIVITY@#
DF      HIGH@YELLOW@INERT@LOW@GREY@HIGH@MEDIUM@
        REDDISH@LOW#
```

The corresponding BNLADS accessing description is

```
PROPERTIES..DOB GOLD,SODIUM,COPPER UNTILB#*
        GETB    DENSITY,A(@);
                COLOR,A(@);
                ACTIVITY,A(@)*
        END PROPERTIES*.
```

As a last example we consider

4) Formatted Mixed Array with Name and Column Label Vector

```
DDF     2300&&&&@TABLE II@@METAL%
        DENSITY%COLOR%ACTIVITY@
        (A($),R4,A(,),R4)#
DF      GOLD$14.8YELLOW,-1.3SODIUM$0.63GREY,4.81
        COPPER$10.6REDDISH,.043#
```

If there are many records in the DF described by the above DDF, the BNLADS accessing description is

```
TABLE II..DØ ⅋ UNTIL ⅋#*
        GET⅋ METAL,A($);
             DENSITY,F4.1;
             COLOR,A(,);
             ACTIVITY,F4.1*
        END ⅋ TABLE II*
```

From the above examples, we can restate the DDF in BNF form relating it to the terminology used to describe the BNLADS accessing description (section 3):

```
⟨ddf⟩::=[⟨control⟩@]
        [⟨external tag⟩@]
        [min[⟨external value⟩%]@]
          1
        [min[⟨recording tag⟩%]@]
          1
        [⟨format⟩]#
```

As a last example, we will consider the hierarchical file structure discussed by Haynes[6] in which the DDF contains five tags describing five different DF accesses:

```
TAG
000     0000&&&&On-site housing directory#
010     2300&&&&&@Apartments@@Apt. No.%Ext.%
         Location@(A(,),I4,A(9))#
020     2300&&&&&@Guest House,13 Upton Road@@
         Room%Ext.@(A(,),I4)#
030     2300&&&&&@Summer Cottages@@No.%Ext.%
         Location@(AC,),I4,A(,))#
040     2300&&&&&@Mobil Homes@@No.%Ext.%
         Location@(AC,),I4,A(12))#
050     2300&&&&&@Residences@@Bldg.%Address%
         Ext.@(A(,),A($),I4)#
```

The accessing description of the BNLADS PLD is

```
On-site housing directory..DØ *
      Apartments..DØ UNTIL 1/13*
            GET  Apt.No,A(,);
                 Ext.,I4;
                 Location,A(9)*
            END  Apartments*
```

```
Guest House, 13 Upton Road..DØ UNTIL 1/13*
        GET Room,A(,);
            Ext.,I4*
        END Guest House, 13 Upton Road*
Summer Cottages..DØ UNTIL 1/13*
        GET No.,A(,);
            Ext.,I4;
            Location,A(,)*
        END Summer Cottages*
Mobile Homes..DØ UNTIL 1/13*
        GET No.,A(,);
            Ext.,I4;
            Location,A(12)*
        END Mobil Homes*
Residences..DØ UNTIL 1/13*
        GET Bldg.,A(,);
            Address,A($);
            Ext.,I4*
        END Residences*
END On-site housing directory*
```

Appendix IV

The ANSI Magnetic Tape Label Format as used by the BNLADS

The BNLADS uses the ANSI magnetic tape label[2] format to describe the sequential files that it accesses. The tags have the following meaning

000	title of file (i.e. PFN, DD)
004	the PLD description
008	old PLD descriptions relating to present file
009	the user's data.

Application programs which construct a new file from old files, list the old PLD in the variable field whose tag is 008. The user's data is a variable data field with tag 009.

The DDL specification of the BNLADS ANSI label is

```
GET  segment control word, A(5);
     subsystem control, A(1);
     character set control, A(1);
     reserved for future use, A(4);
     field control count, A(1);
     base address of data, A(5);
     reserved for future use, A(3);
     entry map, A(4), ANSI Z39.2-1971*
```

```
directory..DØ UNTIL #*
        GET tag,A(3);
            length of field,A(4);
            starting character position,A(5), see PROC
               wrtpld of BNLADS*
        END directory*
padding..DØ UNTIL $*
        GET file character,A(1)*
        END padding*
user's data..DØ *
        GET one char,A(1), user's data considered to
           be in STREAM mode character type *
        END user's data *
```

References

1. Abbey, S., GRIPHOS as a Relational Data Base System, Ph.D. dissertation, Dept. of Computer Science, SUNY at Stony Brook, (1975).

2. ANSI Z39.2-1971 Standard for Bibliographic Interchange on Magnetic Tape.

3. Codd, E. F., A Relational Model for Large Shared Data Banks, CACM 18, (June 1970).

4. Date, C. J., Introduction to Database Systems, Addison-Wesley Publishing Company, (1975).

5. Fuchel, K., Heller, J. and Tischler, J., System 2000 Tape Inventory and Description, memo (1976).

6. Haynes, W., Private Communication, (1976).

7. Heller, J. A Set of Character String Manipulation Procedures in PL/I. Tech Report 14, Dept. of Comp. Sci., Stony Brook, (1972).

8. Heller, J. On Logical Data Organization, Card Catalogs and the GRIPHOS Management Information System, Museum Data Bank Research Report 3, Dec. 1974.

9. Heller, J. and Nardi, J., A Machine Interpretable Design for Physical and Logical Description of Sequentially Archived Data. BNL-20999, (Feb., 1976).

10. Lipka, M. S. E., Queries and Relations in the GRIPHOS System, Ph.D. dissertation, Dept. of Comp. Sci., SUNY at Stony Brook, (1975).

11. Merrill, D. and Austin, D., ERDA Interlaboratory Working Group for Data Exchange (IWGDE), LBL-5329, (Sept., 1976).

12. Nardi, J. Unpublished Report (1976).

13. System 2000 Reference Manual, MRI Systems Corporation (1974).

14. United Nations Headquarters Library Documentation Division, Computer-Assisted Indexing Project, COMP/1, May, 1969.

15. Vance, D., Manual for Museum Computer Network GRIPHOS Application, CCAL Publ. SUNY at Stony Brook, 1976.