DM-1—a generalized data management system

by P. J. DIXON and DR. J. SABLE
Auerbach Corporation
Philadelphia, Pennsylvania

INTRODUCTION

I. Summary of Capability

The basic objective of Data Manager-1 (DM-1) is to provide business organizations with the comprehensive data handling capability outlined in this section.

It is assumed that the types of users will range from those who know nothing about the system and who wish to use it without learning more (e.g., line managers) to those who understand the system well and who wish to manipulate its inner workings to their advantage (e.g., programmers). Consequently, the system has a convenient user-oriented set of languages which shields some users from the complexities of the system.

The following system languages are designed to bring the user as close to DM-1 as possible:

(1) Job Specification Language. DM-1 is capable of storing both generalized system programs and specialized user programs in such a fashion that they can be called upon by a user to be run in the sequence he desires. This job specification capability is effected by means of a command language which is both easy to use and comprehensive in its ability to specify jobs.

(2) Query Language. DM-1 incorporates user-oriented query languages and is designed to incorporate other user-specified languages. It achieves this capability by utilizing an input interpreter which is driven by a syntax table.

Users can initiate conditional searches of arbitrary logical complexity by means of a dialog with DM-1, whereby the user makes a series of simple interrogations each building on the previous one. The products of the dialog culminate in a single complex query.

(3) Data Definition Language. Files or data sets of arbitrary logical complexity can be described and incorporated into DM-1 by means of an easy-to-use data definition language. The manual preparation of data descriptions for even complex data structures is reduced to an almost clerical operation, using elementary work-sheets.

The system does not specify or imply a data structure constrained by the nature of its design, but, rather, it provides a framework for building any reasonable logical data structure desired by the user.

(4) Procedural Language. Users can include their specialized task programs in the DM-1 repertoire by means of a procedural language. The current implementations of DM-1 employ JOVIAL, COBOL, and assembly languages.

All systems exist in an environment of change. The systems that adapt to such changes continue to give useful service. Several adaptive aspects of DM-1 are discussed below:

(1) Modular Design. Additions to the data base can be made once the definition of new data structure has been added to the system directory. New, specialized task programs can be added to the repertoire of the system with no new programming other than that of the task program itself.

(2) Restructuring of Data. In response to a user command, DM-1 is able to extract data from its existing logical structure and rearrange it into a structure more suitable to current use. Furthermore, the system retains statistical indicators of data usage so that it is able to respond to significant changes in usage patterns and to assist the data management personnel in changing the data structure.

In line with the objective of giving maximum convenience to the user, DM-1 is designed to communicate its results back to the user at the earliest possible moment, consistent with reasonable cost. The design objective of DM-1 is to provide the fastest response time available in time-shared multi-user systems by means of console access, where console equipment can be economically justified.

II. Functional Elements of DM-1

Data Manager-1 (DM-1) is designed to provide a
framework within which data may be analyzed, evaluated, summarized, and stored in such a way that it may be retrieved easily and quickly by users rather than data processing specialists. The system elements required to perform this function are the data base, the directories which point to the data, the system programs which use the directories to manipulate the data, and the Supervisor Program which processes the job request and controls the system.

Data base

Since the data base is the basic resource of DM-1, the fundamental strategy of the system is to retain the data in as flexible and accessible a form as possible. The DM-1 data description language not only permits the use of variable length fields and optional items but it also permits nested structures such as the nesting of a variable length file within a record of another file. A further feature is the ability to logically link items that are stored physically with other generically related items, so that a given item may be part of more than one parent item, in the sense of a lattice-type hierarchy. These links are kept in the DM-1 directory so that the identity of all items to be retrieved is known before data access is made.

The data itself is either in random access or magnetic tape storage under the control of the operating system. Data is requested by means of a logical name which does not change with physical data movement. This permits the operating system to change the physical location of the data without having to notify DM-1.

The logical name of a data item is derived from the relative position of the item within the structure of the data base. A unique logical code is created for each item in the data base. The logical code is a numeric representation of the nodes in the tree structure of the data base and is called the Item Position Code. Figure 2-1 shows part of a hypothetical data base. The data items are indicated as branches of a central stem. Each branch is numbered in relation to other branches.

DM-1 supplies a standard language for defining the logical structure of data in the common data base. The language is supplied in two versions. The first uses an indented outline type structure on a standard form to signify the relationships between data items and sub-items. The second uses formal parenthetic punctuation to signify data class relationships and is intended for those users who wish to use a linear parenthetic string input rather than a columnar page format.

Directories

DM-1 provides the user with a data description language which permits him to specify a very wide range of data structures. The structural description of the data is implemented by means of the DM-1 directories which tie the system to the data base.

The data can be recalled to core memory only by means of system directories. A function of the directories is to translate the names of data items, first, into logical codes which describe the relative or logical positions of the items and, second, into the symbolic names of the data segment which can be used by the operating system to fetch the data. All system functions of file maintenance and retrieval depend on the directories to locate the desired data and to describe it once it has been found. The four main directories to the system are described in the following paragraphs.

Figure 2-2 shows what each directory requires as input and what each is designed to provide. The functions of the data directories are as follows:

(1) They are utilized to focus in on the data.
(2) They are utilized to give a description of the data in the record (e.g., whether the data is a floating point or an alphanumeric number).
(3) They are used to extract the data.
(4) They contain the index values for the data items so that searches can be performed within the directories without having to access the data.

The main directory tables are:

(1) Term Encoding Table. The basic function of the Term Encoding Table (TET) is to convert the name of a data item from its alphanumeric input form to a coded form which describes the logical position of the item in the data structure. The coded form, called an

![Figure 2-1—Structure of purchasing data base](From the collection of the Computer History Museum (www.computerhistory.org))
Figure 2-2—Directory inputs and outputs

Item Class Code (ICC), consists of integers which represent the nodes on the tree structure describing the data. For example, the ICC for Order List File in Figure 2-1 would be 1.3.R.A. A given term will have more than one ICC associated with it, if it is an item used in the data structure more than once.

(2) Item Position Index. The Item Class Code must be converted into a unique code by supplying values for the R's before a specific item can be retrieved from random-access storage. The conversion is performed by the Item Position Index, which is arranged by ICC and which contains all of the values by which certain data items have been indexed. When the ICC has been changed into a unique set of integers with no R values, it is called an Item Position Code (IPC).

Another function of the Item Position Index is to retain the statistical tallies of data usage, since all usage must pass through the Item Position Index. It is from these tallies that the need for data restructuring will become evident.

(3) Segment Name List. An Item Position Code is a code for a unique data item. It embodies sufficient information to be able to call for its appropriate data segment. Determining the name of the segment to be called is the task of the Segment Name List. When the name has been determined, it is given to the I/O Control Program along with a request for the segment, and the I/O Control Program retrieves the segment from peripheral storage and places it in a prescribed input area in core memory where the exact data item can be located.

(4) Segment Index. The final step in retrieving a data item is taken with the aid of the Segment Index. Each segment of data begins with a Segment Index which points to the data items contained in the segment. The Segment Index uses the IPC and auxiliary information from the Item Position Index to find its way to the desired item. At this point, there are several things which may be done with the data. The item may simply be retrieved, deleted from the segment, or used as a foundation upon which to add new data items to the file.

Job specification

A job is defined as a sequence of program tasks that accomplish some desired function for a user. DM-1 is designed to perform a variety of different jobs with a minimum of programming through the use of many generalized programs. The user will be able to call on these programs in the sequence he desires.

The user accomplishes this variety of uses by means of a Job Request. The Job Request performs the functions of item definition, data entry, program entry, job entry, and job running. The first four Job Request types perform system maintenance functions. The fifth Job Request type permits the user to run jobs of his own on the system.

A standard command language is provided for the user to specify each job he wishes accomplished.

Job specification resources afford both a dynamic and modular expansion to DM-1. New data can be added to the data base in batches or on-line, with few constraints on the type of data. Modular additions can be made to the system functions to incorporate new task-oriented programs. The Job Request types are summarized below:

(1) Item Definition Request. The logical structure of the data base can be altered through an Item Definition Request. An alteration is made in the directory entry which describes the relationship of the data item
to the data base. Possible alterations include the addition or deletion of data items such as statements, files, records, or fields.

(2) Data Entry Request. Data may be added or deleted by issuing a Data Entry Request. This type of request would normally be used for transactions where the data in the data base do not have to be examined in order to complete the transaction. More complex transactions may be effected by combining the Job Entry Request and Job Run Request which are described below. Data may be entered by means of a console or by magnetic tape. The data may be in the internal format of the system, or it may be in the external format of the data-description language.

(3) Program Entry Request. Descriptions of programs are entered into the system with the Program Entry Request. The programs and the descriptions of their parameters are given a unique name, and the descriptions are placed in the Program Description List where they are on call for the execution of jobs at a later time. Program Descriptions are used for checking purposes to be certain that subsequent jobs, which call on the programs, do so correctly, with the proper parameter specifications.

(4) Job Entry Request. The Job Entry Request is used to bring a Job Description into the system. A Job Description specifies the program to be run for a job and the sequence for running the programs. Any predetermined program parameters for the job may also be entered by this means. A Job Description is usually entered for a job which is run frequently, so that the job does not have to be specified each time it is run. A Job Entry Request causes a Job Description to be stored so that it may be called later by a Job Run Request. Upon entry, Job Descriptions are checked carefully by the system to be certain that they are compatible with the requirements of the programs they use.

(5) Job Run Request. A Job Run Request asks for a particular job to be run. A job must have been described previously in a Job Description by means of a Job Entry Request. If the Job Description specifies all parameters, the Job Run Request can be so brief as to simply identify the job name to be run. However, if the Job Description refers to several programs requiring parameters which have not been pre-assigned in the Job Description, the Job Run Request must specify these parameters in detail.

System programs

The programs in the system consist of two types:
(1) System-Oriented Programs. The programs are relatively stable and they provide the basic framework for processing data and for implementing other programs. They constitute the underlying programming foundation upon which all other system tasks are based. Examples of system-oriented programs are those which scan input data, access and manipulate system directories, locate data in the data base, and do routine jobs such as sorting and merging.

(2) Task-Oriented Programs. These programs are components of user jobs. They use the system-oriented programs to help them accomplish the required tasks. The task-oriented programs are less stable, in that they may be changed to suit the needs of an evolving task or the user easier than the system-oriented programs. That is, they may be added or deleted with much less effect on the system. Examples of task-oriented programs are those which summarize and analyze data and those which generate specialized reports.

The two types of programs are combined by a Job Description and work together to accomplish useful work. The data manager or systems analyst who is responsible for accomplishing a task creates a Job Description through a Job Entry Request. Although many system-oriented programs concern themselves with systems maintenance only, many others can be used as subroutines in user-oriented programs for sorting, merging, extracting, formatting, etc.

The use of the two program types may be illustrated by the following example. A technical specialist may wish to extract some prescribed fields of data from a permanent file in order to build a temporary file which is more suited to his purposes. He may plan to perform a series of analyses on the new file and have the results reported in a certain order and format. The specialist would issue a set of Job Entry Requests to produce the necessary Job Descriptions to tell the system what to do. The Job Descriptions would consist of a list naming both system-oriented and task-oriented programs. The system-oriented programs would scan the inputs which have been written in a user language, locate and extract data from the old file, construct the new file structure, and extract from the new file those data items desired by the analysis routines. If a generalized report generator were used, it too would be system-oriented, in that it would be available to all system users. The task-oriented programs would permit the system to select the data according to a Boolean logical condition, summarize the data, analyze it, and prepare the desired output if a specialized report routine were used.

System-oriented programs which deserve special mention are the Filesearch and the Table Access Packages. These generalized programs are responsible for all basic data manipulations and searches. They use the directories for extracting, storing, and altering data in the data base and for restructuring data into a
new form. One of the prime features of these programs is that they can make full use of the index data found in the Item Position Index to perform conditional searches of the data base. Further, they can manipulate the index data and reduce to a minimum the number of accesses to peripheral storage. This feature provides the foundation for accepting a query in any defined query language, and for processing it in an efficient manner.

III. Directories and the data base

A system—which efficiently processes data stored on random-access devices must rely heavily on directories to keep track of the data in the system. The directories, and the programs which utilize them, are critical elements of the system because they constitute the retrieval mechanism of the system and are used with great frequency during data manipulation.

The interaction between the directories and the data base is particularly important for two reasons:

(1) DM-1 directories are capable of being either specific or generic when they point to data. In contrast, the directories of many random-access data systems can be generic only, since they point to a relatively large group of data for retrieval and then scan it in core memory to determine which DM-1 part of it satisfies the search conditions. DM-1 can search in this generic way also, and since the DM-1 directories point to specific items, they can isolate small data items (e.g., fields or records of files imbedded in other files) which satisfy the search conditions before actually retrieving the data. This capability facilitates searching and increases the importance and utilization of the directories.

(2) DM-1 directories are used to describe the logical format of the data. Since DM-1 is sufficiently flexible to permit almost any kind of logical data format and structure, it must have a means of cataloging the various structures contained in the data base. The directories provide the “template” necessary to describe these structures, thus making the directories an indispensable part of the data handling process.

The following paragraphs describe the nature of the DM-1 directories by taking the reader through each functional step required to locate a particular item of data in the data base. This simple search process is only one of many uses for the directories, but it will serve to acquaint the reader with the basic mechanism for their use.

For the purpose of this description, it will be assumed that a user has entered a Job Request into the system to find a data record of a certain type. Although the case is artificial, let us assume that the user wants the first ORDER record in the ORDER file which represents a purchase order for J. Jones. The user knows there is a field in each ORDER record called REQUESTER, and he wants the system to search the ORDER file until it comes across a record with a value in the REQUESTER field which is J. JONES. The part of the Job Request that would describe this record (and others like it) might look like this.

ORDER, REQUESTER = J. JONES

The system would know that it must focus in on the item called ORDER and that it must select the first record which meets the criterion of having the value J. JONES in the field called REQUESTER. The following paragraphs describe how the selection is accomplished.

Term Encoding Table. The first step in locating the record is searching the Term Encoding Table for the Item Class Code (ICC) associated with the record. The prime parameter for searching the Term Encoding Table is a specific alphanumeric name of a data item called a term. If the term is unique in the system, it can be used alone. If the term is used to name items in more than one file, the term must be accompanied by terms higher in the data structure which give it sufficient context to make it unique. For example, Figure 2-1 presented a tree structure which included three occurrences of the term VENDOR No., in the records of the VENDOR, ORDER, and ITEM files. One must precede the term VENDOR No. by the term VENDOR, ORDER, or ITEM to make VENDOR No. refer to a unique item.

The Term Encoding Table is reached by means of the Term Encoding Table Directory. This Directory (see Figure 3-1) is brought into core memory, if it is not already there. The Directory contains an entry for each data segment of the Term Encoding Table. The argument of the Directory consists of the alphanumeric names of the first terms in each segment, arranged alphabetically. The function values of the entries are the names assigned to the segments by the I/O Control System.

The segment of the Term Encoding Table which contains the entry for REQUESTER is brought into core. The retrieval is performed by matching REQUESTER against the argument of the Directory.

<table>
<thead>
<tr>
<th>ARGUMENT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTS</td>
<td>1602591</td>
</tr>
<tr>
<td>DUE DATE</td>
<td>7679729</td>
</tr>
<tr>
<td>PRICE</td>
<td>4796215</td>
</tr>
<tr>
<td>VALUE</td>
<td>2387224</td>
</tr>
</tbody>
</table>

Figure 3-1—Term encoding table directory
For example, if the Term Encoding Table is four segments long, the Directory would have only four entries, as in Figure 3-1. REQUESTER is found to lie between PRICE and VALUE in the argument, hence, the segment of the Term Encoding Table which contains information about function starts with the term PRICE. The system might call this segment 4790215 (its symbolic name). Segment 4790215 is retrieved by the I/O Control Program.

The Term Encoding Table is almost as simple as its Directory but is much larger. The argument contains all the term names used in the system, arranged in alphabetical order. Figure 3-2 illustrates part of a hypothetical Term Encoding Table which includes most of the terms in the tree structure of Figure 2-1. The function consists of the appropriate Item Class Code for the term name. The ICC is derived by tracing the nodes in the tree structure to the item being coded. As mentioned previously, some terms will name more than one item within the tree structure, and a function entry will occur for each appearance of the name. For example, ITEM No. appears in both the ITEM File and the ITEM LIST File of the ORDER File and, hence, has two ICC entries in the function of the Term Encoding Table for the codes of the two item classes.

The Term REQUESTER is looked up in the argument of the Term Encoding Table. If only the ICC entry is found for REQUESTER, the item name is unique within the data structure, and the single ICC may be used. If more than one ICC entry is found, as in this case (see the bottom arrow in Figure 3-2), the desired context of the item must be established. The context is derived from the input term which precedes the term being processed. In this case, the term ORDER precedes the term REQUESTER in the input query. ORDER is found to have an ICC of 1.2 (see top arrow in Figure 3-2). The root of 1.2 is the proper context of REQUESTER, and 1.2.R.3 is selected as the correct ICC rather than 1.3.R.4.R.2. If the preceding term is insufficient to establish the proper context, the next preceding term is used until either the context is established or it is found that more input terms are required.

The primary output of the Term Encoding Table is a single ICC (in this case 1.2.R.3). Other outputs may consist of error messages if a term cannot be found in the Term Encoding Table, or if there are not enough terms to establish a unique context.

**Item position index**

The next step is to use the Item Position Index to convert the Item Class Code into an Item Position Code, thereby establishing a unique logical address for the data items. Two kinds of parameters are associated with the Item Position Index. The most important parameter is an Item Class Code. If the ICC represents a unique item instead of a class of items (i.e., there is no R in the ICC), no other parameters are required.

If the ICC contains the letter R, a further parameter must be supplied in order for the Item Position Index to convert the ICC into an IPC. The parameter may be a value for a field which has been indexed, such as in the present case where the value in the field REQUESTER has been designated as J. JONES. On the other hand, the parameter may be simply to obtain the first record in the file, or the next record in the file.

The Item Position Index is reached by means of the Item Position Index Directory in exactly the same fashion as the Term Encoding Table has been reached by the Term Encoding Table Directory. As seen in Figure 3-3, the argument of the Directory consists of ICC's arranged in alphanumeric order. The segment of the Item Position Index which contains information about 1.2.R.3. is found to start with 1.2, and the Segment Name is 4287933 (see the arrow in Figure 3-3).

The Item Position Index has an argument similar to its Directory: ICC’s arranged alphanumerically. However, the argument of the Item Position Index contains all ICC's which are used in the system and is relatively complex, as seen by Figure 3-4. Note that this figure depicts only the logical relationships between portions of data in the directory but gives no indication of the
Figure 3-3—Item position index directory

physical location of the data. For instance, the Index Value data and R Value data will be in subordinate directories of their own, linked to the main Item Position Index by addresses.

<table>
<thead>
<tr>
<th>ARGUMENT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Name of First ICC in Segment)</td>
<td>(Segment Name)</td>
</tr>
<tr>
<td>0</td>
<td>5446932</td>
</tr>
<tr>
<td>1, 2, 3, 4, 5</td>
<td>8746951</td>
</tr>
<tr>
<td>1, 2</td>
<td>1297213</td>
</tr>
<tr>
<td>1, 3</td>
<td>7434654</td>
</tr>
<tr>
<td>1, 3, 4, 5</td>
<td>4062225</td>
</tr>
<tr>
<td>1, 2, 4, 5, 6</td>
<td>9946663</td>
</tr>
<tr>
<td>2, 3, 4, 5, 6</td>
<td>729261</td>
</tr>
<tr>
<td>2, 3, 4, 5, 6, 7</td>
<td>2321714</td>
</tr>
<tr>
<td>2, 3, 4, 5, 6, 7, 8</td>
<td>6444980</td>
</tr>
</tbody>
</table>

Figure 3-4—Item position index

Figure 3-4 shows the logical relationships between entries in the Item Position Index. For each ICC in the argument, there are the following types of associated information:

(1) **Term**. The term entry after the ICC makes the Item Position Index the inverse table of the Term Encoding Table and permits the conversion of an ICC back to its original alphanumeric term. This capability is particularly important in man-machine communications where machine codes must be translated before being presented to human operators. In the actual design, the term entry is a link address back to its associated entry in the Term Encoding Table, rather than a repetition of the term name itself. This technique saves space while losing no efficiency.

(2) **Item Type**. An ICC can designate an item class of several types, and the code for the type is entered in this part of the function. The most generic item type is a **statement** (e.g., PURCHASING INFORMATION), where the type refers to no specific entity of data. Another type is a **file** (e.g., ORDER File; 1.2), which contains any number of records. A **record** is a type having exactly the same logical structure as other records in the same file (e.g., ORDER Record; 1.2.R). A record may contain other files or it may contain fields of three kinds. A **variable** field (e.g., REQUESTER; 1.2.R.3) is one whose length changes according to the length of the data in the field. A **fixed field** (e.g., DUE DATE; 1.2.R.2) is always the same length, and the length in characters
is indicated in the function of the Item Position Index. The Item Type also specifies the mode in which the data of a field is stored, such as binary, octal, integer, decimal, floating point, or alphanumeric.

(3) **Index Values.** It is anticipated that many data fields will be indexed, thereby permitting searches for specific fields without having to access large amounts of data. When a field in a series of records is indexed, all the different field values in the records are listed under Index Values in the Item Position Index. If values are repeated, they need be listed only once. Reference can be made to the values without having to retrieve the records. An example in Figure 3-4 lists the field value range of zero to 1300 under the field called PO No.

(4) **Index Code.** Three indexing options are available to a program performing a file updating function. One option is to index all values which occur in a given field, whether similar values have been indexed or not (e.g., ALL, after PO No.; 1.2.R.1). Another option is to continue to index only those values which have already been indexed in the past (e.g., SAME, after DUE DATE; 1.2.R.2). The final option is to not index the field at all (e.g., NONE, after ITEM No.; 1.2.R.4.R.1).

(5) **R Values.** To give the indexing meaning, the index values must be directly related to specific fields. The relationship is provided by means of R Values which can be substituted for the letter R in the ICC. For example, Figure 3-4 shows that the term DUE DATE has an ICC of 1.2.R.2 and that it is an indexed fixed field. If one wanted to identify an item in this class which contained the value FEB 66, any of the R Values 1, 2, 3, 8 or 9 could be substituted in the ICC 1.2.R.2 for R to give a unique series of integers which would identify the exact relative (or logical) position of such a field in the data structure. The series of integers, with no letter R contained in it, is called an Item Position Code (IPC).

To demonstrate the use of the Item Position Index in locating specific fields of data, let us return to the example started at the beginning of this section. The ICC obtained from the Term Encoding Table is 1.2.R.3, and the field desired is called REQUESTER. The value of the field is specified by the Job Request as being equal to J. JONES. Assume that the portion of the Item Position Index given in Figure 3-4 is equal to a segment. The program searches the argument for the ICC equal to 1.2.R.3 and checks the term for consistency. The program notices that REQUESTER is a variable-length field which is indexed. It searches down the list of Index Values for J. JONES. When J. JONES is found, the R Value associated with J. JONES is extracted (in this case, the integer 6) and is substituted in the R of the record ICC to form the IPC 1.2.6. If more than one R Value exists, the other values can be used in subsequent searches for more transistor records having REQUESTER fields with the value J. JONES. The important fact to notice is that the IPC is now unique within the whole DM-1 data structure, and it may now be used to point to the exact data record desired by the user. (The IPC could have been used to point to a specific field or bit in the record.)

The problem of conducting Boolean searches for records meeting certain criteria can now be discussed with some understanding of the mechanism by which the searches may be conducted. Suppose that a Job Request wants all Order Records which have a purchase order number less than 1300, which are due in February, 1966, and which were requested by Mr. A. Smith. The program can examine the three fields, PO No., DUE DATE, and REQUESTER, for the desired Index Values. The list of R Values corresponding to these Index Values can be extracted and compared for any integers in common. In the case shown by Figure 3-4, the integer 2 is common to all three desired values. Hence, the R of ICC 1.2.R. can be replaced by 2, and the record identified by IPC 1.2.2 may be retrieved from random-access storage with full assurance that it meets the criteria of the Job Request Query. In this manner, the Item Position Index can be manipulated to determine what records are desired without having to fetch from storage any more records than those which exactly fit the criteria.

DM-1 files can be dynamic, with constant addition, deletion, and change. The Item Position Index is the logical place to keep track of how many records are contained in each file and of what gaps may exist in the logical structure of the file. Normally, an entry occurs in the R Value column of the Item Position Index only for indexed fields, but record fields are an exception. Figure 3-4 shows three integers in the R Values portion of the function for ORDER (1.2.R). The last integer (12) gives the R Value which should be used for the next record added to the file. Thus, the number of records in the file may be calculated by subtracting one from the last integer. The first (i.e., 5) and the second (i.e., 10) integers indicate R Values which were once part of the file but which have been deleted and not reused. New additions to the file can receive R Values for their IPC’s by means of the entries in this column.

Two sets of statistics are retained in the Item Position Index to keep track of system usage. The first is a tally kept for each time a field value is used in a conditional search. Periodic analyses of these tallies permit the data base manager to unindex those values that are no longer used.
which are seldom used, thereby conserving storage space. The second is a tally kept for each time a field is accessed, whether conditionally or as a result of serial processing. These tallies help to show the usage relationships between data items and aid the data base manager when he restructures data for more efficient processing.

When files are placed within the records of other files, multiple R's are found in the ICC's, and some complexities arise. For instance, a single set of integers is not sufficient to show the R Values for a single ICC, because the ICC now has multiple R's in it. Since each R represents a multiple number of records, the possible number of integer sets grows exponentially with each new R in the ICC. The problem of organizing the sets of R Value integers can be solved by linking with addresses each integer of the first set to its appropriate second level set, and so on down the hierarchy of R levels in the ICC.

There are several outputs which can be derived from the Item Position Index. The Item Position Code is the major output because it may be used to point directly to the data field being sought. The Item Type, another output, is also important because it describes the nature of the field being dealt with, such as the field length in the case of a fixed field.

A number of possible error conditions may be encountered while manipulating the Item Position Index: the desired IPC may not exist in the argument; the term in the Item Position Index may not agree with its counterpart in the Term Encoding Table; the field, for which a value was given on input, may be found to be unindexed; and many other possible internal inconsistencies. The output from the error conditions will depend entirely on the operating policy of the system at the time of implementation.

In this case, the output is an IPC with the value 1.2.6.

**Segment name list**

Looking up the name of the segment containing the desired data is the next step in locating the desired ORDER record. The names are kept in the Segment Name List. The parameter required to use the Segment Name List is a single IPC, which is used to request from the I/O Control Program the segment which contains the desired item.

As the data base grows, the Segment Name List may become large enough to require its own Directory. The Segment Name List Directory will be brought into core memory, just as the Term Encoding Table was, selecting the segment with the next lower argument closest to the IPC being sought.

<table>
<thead>
<tr>
<th>ARGUMENT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>First IPC in Each Segment of the Segment Name List</td>
<td>(Segment Name)</td>
</tr>
<tr>
<td>0</td>
<td>4968431</td>
</tr>
<tr>
<td>1.3.13.7.2</td>
<td>584268328</td>
</tr>
<tr>
<td>2.2.4.24.6</td>
<td>9206307</td>
</tr>
</tbody>
</table>

**SEGMENT NAME LIST DIRECTORY**

<table>
<thead>
<tr>
<th>ARGUMENT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>First IPC in Each Segment of the Data Base</td>
<td>(Segment Name)</td>
</tr>
<tr>
<td>0</td>
<td>6964320</td>
</tr>
<tr>
<td>1.1.2.5</td>
<td>7032694</td>
</tr>
<tr>
<td>1.2.4.1</td>
<td>8268215</td>
</tr>
<tr>
<td>1.2.11.4.19.3</td>
<td>4394762</td>
</tr>
<tr>
<td>1.3.7.4</td>
<td>1147236</td>
</tr>
<tr>
<td>1.3.94.1</td>
<td>6468945</td>
</tr>
<tr>
<td>1.3.112.6.1</td>
<td>2325464</td>
</tr>
</tbody>
</table>

**SEGMENT NAME LIST**

The Segment Name List is exactly like the Segment Name List Directory, only much larger (see Figure 3-5). Whereas the Directory argument has the first IPC of each Segment Name List segment, the Segment Name List argument has the first IPC of each data base segment. In essence, the Segment Name List is an extension of its own directory. Only directories can be ordered from the Local Control Program without using the Segment Name List, and this ordering is done through a directory which, in reality, is a small Segment Name List.

In our test case, the IPC desired is 1.2.6. The routine orders the first segment of the Segment Name List with the Segment Name of 4968431 (see the top arrow in Figure 3-5). The Segment Name List Segment brought into core memory looks something like the bottom table in Figure 3-5. The Segment Name List is used to select the data segment which contains IPC 1.2.6. In this case, the segment begins with IPC 1.2.4.1 and has a Segment Name of 8268215 (see the bottom arrow). The Segment Name is transmitted to the I/O Control Program, and an interpretive request is given the I/O Control Program to retrieve the data segment.

The Segment Name List also contains a Usage field to hold data on the number of times the segment has been retrieved in the last statistical data period. Analysis of this data will permit optimal use of the various storage media of the computer center.
Segment index

The final step in locating the data is made with the help of the Segment Index at the head of the segment containing the data. The major parameter for using the Segment Index is the IPC which is being sought. Another parameter is the Item Type (showing the length of the field, if it is a fixed field). A third parameter shows the beginning address of the segment in core memory. In actuality, the portion of the Item Position Index already referred to is another parameter, because the Segment Index and Item Position Index are used in conjunction with each other to locate the data item. A segment will be divided into four parts: a heading, a segment index, data, and vacant space. Except for the heading, the proportion of each part will vary, depending on how much data exist in the block and on how much indexing data are required to access the data.

To illustrate how the search would proceed, let us first consider what the string of data in the recently retrieved segment might look like. Figure 3-6 uses some symbols to show the various types of items and their logical relationship in the data structure. Although the symbols are shown on four lines, the items should be considered as one continuous string of data. The IPC for each item has been posted under the item, for visual reference. It will be noticed that the data string begins the segment and that the IPC value of 1.2.4.1 corresponds with the starting IPC of the segment just retrieved by means of the Segment Name List.

The DM-1 design specifies that the Segment Index consists of a list of addresses showing the starting position of data fields in the segment. Figure 3-7 shows a sample Segment Index. The first address always shows the start of the data area (i.e., address A) and is relative to the beginning of the segment. The following addresses (i.e., B-F) point to subsequent data fields and are relative to the beginning of the data area. The final address in the Segment Index (i.e., address G) is always the end of the data area or the beginning of the vacant area. However, only those addresses that are vital to the process of locating data are included in the Segment Index. A series of fixed fields, for instance, needs only the starting address of the series, because the starting addresses of the rest of the fixed fields in the series can be calculated from the Item Position Index. Normally, only the end of variable or optional fields need be indicated. In Figure 3-6, the IPC addresses needed to scan the data have been circled. The other addresses need not be carried in any index, and it will be noticed that in Figure 3-7 only the circled addresses have been included in the Segment Index.

The basic logic for locating data in a segment is to use the Item Position Index to calculate where the desired address is in the Segment Index. The program starts with the ICC in the Item Position Index which corresponds to the IPC at the start of the segment. The program steps through the Item Position Index, while keeping count of how many addresses there should be.
in the Segment Index up to that point. When it gets
to the desired IPC, the program goes directly to the
Segment Index and counts down through the index the
same number of addresses to the desired address. This
address points to the data with the specified IPC,
thereby completing the retrieval function.

Normally, the program need go to the segment
itself only once to fetch the data. If an optional field
is discovered in the Item Position Index, however, the
program must go to the data to see if the field actually
exists.

To see how the process operates, let us follow the
logic of how the data for IPC 1.2.6 is retrieved. The
program knows from the Segment Name List Di-
rectory that the segment begins with IPC 1.2.4.1 and
that the first address of the Segment Index will point
to that data field. The program consults the Item
Position Index for the Item Class Code which cor-
responds with 1.2.4.1. It is found to be 1.2.R.1. The
R Values for the file ORDER show that the fourth
record is a legitimate member of the file because no
4 appears under R Values to show that the fourth
record is missing. The program notices that the first two
fields are fixed, followed by a variable field, and the
program deduces that a second address in the Segment
Index will be required to point to the end of the
variable field called REQUESTER or to the beginning
of a fixed field called VENDOR No. Further, the
program deduces that a third address will be required
to point to the end of a variable field called VALUE
or the beginning of a possible file called ITEM LIST.
A link to a subsidiary list of second level R, Values (not
shown in Figure 3-4) supplies the information that
only one record exists in the ITEM LIST File when
the first R in the ICC is equal to 4. The program
deduces that a fourth address in the Segment Index
must indicate the end of the QUANTITY field and the
beginning of the COST field (IPC 1.2.4.4.1.3). A fifth
address is needed to signal the end of the COST field,
because it is variable, and the end of that inner file and
record. It is also known that the fifth address skips to
the beginning of record 1.2.6, because the Item
Position Index R Values for ICC 1.2.R. state that record
1.2.5. does not exist in the file.

Since IPC 1.2.6 represents the desired record, the
program can now turn to the Segment Index and
select the fifth address in the index. The program can
then go to the data location in the segment which has that
address, and the program will be positioned at the
start of the desired record. The end of the record
can be determined by finding the address which starts
the next record, using the same procedure as described
above. The intervening data may be retrieved, thereby
completing the search process.

IV. DM-1 job requests

General

An important feature of DM-1 is its ability to respond
to Job Requests, either immediately upon entry or
after they have been stored in the system for a period
of time. Much of the operational power of DM-1
is achieved by permitting users to sequence and link
generalized programs by means of Job Requests. The
remainder of this section covers a general concept of
what happens within the system when each of the five
job types is initiated.

Item definition request

An Item Definition Request can be used either to
add or to delete the definition of items in the file
structure. Changes are affected by means of a deletion
followed by an addition which represents the correct
version. The input to the system from an Item Defini-
tion Request which adds a definition will originate on
a standard input form. The user will show the relations-
ships between items and subitems by the use of in-
dentations on the form, which may be tabbed. The item
definitions will be keypunched to show the number
of indentations for each entry.

Upon receipt of the definition data, the Supervisor
program begins executing the program in the Add
Item Definition job. The programs step through the
terms of the input and note the logical relationships
between terms. A series of records is created contain-
ing each term and its associated interim tree code,
along with other information intended for inclusion in
the Item Position Index, such as indexing code or
Fixed-Field Length. The items in the records will
automatically fall into order by tree code, because the
input of the Item Definition Job Request would be in
hierarchical order. The base node is checked to be
sure it exists in the directory and that it can be used
as a foundation for further expansion.

Separate records are prepared for each of the items.
They are sorted by term name for the purpose of up-
dating the Term Encoding Table. The original set of
records is in ICC order, and it is used to update the
Item Position Index by means of a single insertion,
since the ICCs are in a cluster. A generalized print
routine can be called upon to print out both series of
records for manual accuracy checking and for updating
manual records of the Term Encoding Table and the
Item Position Index.

For deletions to the directories, a different process
is employed. The last item in the input of the Job
Request names the item to be deleted. In addition, all
other items which are subsumed by the named item
are to be deleted. The Term Encoding Table and the
Item Position Index are used to create the ICC of the item named for deletion. The entry in the Item Position Index is inspected for the item to be deleted and for all lower items in the same tree code branch (i.e., having the same ICC root). If the index shows that data exist in the database for any of these items, an error condition is considered to exist because the definition for existing data must not be deleted. Data must not be permitted to exist in the database without being properly defined. Thus, if data exist, the definition is not deleted, and a printout is prepared for the person initiating the Job Request instructing him to delete the data first if he wishes to delete the item definition.

If no corresponding data are found to be associated with the named item and its subsidiary items, the entries for the items are deleted from the Item Position Index. Pertinent items are deleted from the Term Encoding table, and a printed audit trail of deleted items is prepared.

Data entry request

The Data Entry Request may be used either to add data to or to delete data from the database. With additions, the data are not converted from one logical structure to another, but they are translated from one physical structure to the IMS physical structure so that it can be incorporated into the database. Data may be added in any one of four modes, as specified by the Data Entry Request (see the top half of Figure 4-1). The four modes are as follows:

1. Defined-External Mode. This mode requires that an Item Definition, describing the data to be entered, has already been entered into the system and has been incorporated in the system directory. The data are entered in external format; that is, the data are manually formatted and named by a data specialist using a simple input worksheet.

2. Undefined-External Mode. This mode has the same external format as the Defined-External Mode, but it does not contain a definition of the data being entered. Consequently, the definitions, in the form of an item image, must precede the data on the input medium.

3. Defined-Internal Mode. This mode requires that a complete definition of the data be included in the system directory before the data are entered, just as in the Defined-External Mode. The format of the data, however, is different. The internal format is defined as being synonymous with the segment format used in the data segments of DM-l. Internally formatted data come in increments of a segment and have their own segment index at the beginning of each segment. The data are entirely dependent on the system directory for data identification and processing.

4. Undefined-Internal Mode. The last of the four modes does not require that the system contain the definition of the data to be entered, because, like the Undefined-External Mode, it contains its own definition. The data are in internal organization but the data also have a local Term Encoding Table and Item Position Index heading the blocks of data on the tape.

Program entry request

Compiled program code will be entered into the system according to established local conventions, and these programs will be called at execution time by means of the IOCS. Descriptions of the programs, however, are entered into the DM-1 directory for the purpose of inventory control and for internal checking to be certain that Job Descriptions are compatible with the programs they call on. The task of the Program Entry Request is to add or delete a Program Description from the Program Description List.

For an example of how a Program Entry Request might look, the reader is referred to the top portion of Figure 4-2. The example says, in English, "I want to enter two Program Descriptions into the system. My name is Jones and my identification number is 87. The deadline for entering the program is 9:30 AM on February 13. I wish to add to the Program Description List, and the Program Description will follow on the console. The first program I wish to enter is called ORDER SEARCH. The first input parameter of ORDER SEARCH is called REQUESTER, and REQUESTER must be an alphanumeric which is of variable length.
The second input parameter of ORDER SEARCH is called VENDOR, and VENDOR must be an alphanumeric which is ten characters long. There is only one result (i.e., output) parameter for ORDER SEARCH, and it is a file called SEARCH PRODUCT. The second program I wish to enter is called VALUE ANALYSIS and it has only one input parameter which is a file called ANALYSIS DATA. VALUE ANALYSIS has only one result parameter, and it is a file called ANALYSIS PRODUCT.

The basic functions of the Program Entry Request Routine may be summarized as follows:

1. To associate a name with a program so that it may be called at a later time.
2. To define the prerequisite parameters of a program so that the parameters given in any program description which calls for the program may be checked for format and completeness before the program is run.

Job entry request

The basic functions of the Job Entry Request are as follows:

1. To check the consistency, accuracy, and completeness of an incoming Job Description by comparing it with the Program Descriptions to which the Job Description has referred.
2. To merge a valid Job Description with the Job Description List so that it may be used at a later time for running a job.

The basic functions of a Job Description are as follows:

1. To name the programs which must be run in order to accomplish the task assigned to the job.
2. To specify the sequence in which the programs should be run.
3. To specify the values or names required by the input parameters of the program to be used in the job, or to specify that the values or names will be given by the Job Run Request at a later time.
4. To specify the names of portions of data resulting from the program so that these output parameters may be referred to and used by subsequent programs that have the same names specified in their input parameters.
5. To specify the names of multiple program exits so that programs may be linked by the Job Manager of the Supervisor Routine according to various contingencies.

The Job Entry Request provides the raw data for a Job Description. An example of a Job Entry Request may be seen in Figure 4-2. Aside from the usual header information, the request says, “I would like to enter a Job Description into the system via the console.” The job is called JOB ONE, and it calls for the running of two programs. The first program is called ORDER SEARCH. I do not wish to give a value at the present time to the first input parameter of ORDER SEARCH called REQUESTER. The value of TYPE will be named at execution time (when a Job Run Request calls for the job), so I will give it the job parameter name of ALPHA, and I will put an asterisk in front of the name to distinguish it from a value. I wish to assign the value RCA to the second input parameter called SEARCH PRODUCT. The second program is called VALUE ANALYSIS. I want to assign the name OUT to the input parameter, which the program VALUE ANALYSIS has called ANALYSIS DATA. (This assignment will connect it to the result parameter of the program.) I assign the name TOTAL VALUE to the job result parameter, which the program VALUE ANALYSIS has called ANALYSIS PRODUCT. Since a value has not been supplied to the job input parameter called ALPHA (the program parameter called REQUESTER), I will place a description of the parameter requirements at the end of the Job Description for reference independent of the program. The requirement is that the parameter be a six-character alphanumeric.
programs referred to by a Job Description be entered into the system before the Job Entry Request is processed. Processing takes the following form:

1. The Program Descriptions associated with the Job Description are retrieved from the Program Description List. Any missing programs signal an error condition which terminates the routine. The Job Descriptions of both the Program ORDER SEARCH and the Program VALUE ANALYSIS are brought into core memory.

2. The input and results parameters given in the Job Description are compared with their counterparts in the Program Descriptions to ensure that all parameters have been accounted for.

3. At the same time, the limitations imposed on the parameters by the Program Descriptions are checked against the nature and format of the parameters given by the Job Description. For instance, the second parameter of the Program ORDER SEARCH, called VENDOR, must be an alphanumeric because the letter A is given. On checking the Job Description for the Program ORDER SEARCH, the routine finds the value RCA given. The value is an alphanumeric and satisfies the limitation. A number of the subroutines used to perform the above two tasks can be shared between the Program Entry Request Routine and the Job Entry Request Routine.

4. One of the most important tasks of the routine is to scan the Job Description parameters for consistency to be certain that the result parameters specified match the input parameter requirements. For example, the result parameter, SEARCH PRODUCT, from Program ORDER SEARCH is called OUT. The routine scans for a matching input parameter which, in this case, is the parameter, ANALYSIS DATA, of the Program VALUE ANALYSIS. Once the match is determined, the limitations are compared. Any result or input parameters which require mates, but which do not have them, or which have improperly written mates, are printed out for subsequent correction.

5. Any parameters which are unspecified at the time of Job Description entry are formatted at the end of the description for easy checking when they are specified by a Job Run Request.

Once the Job Description has been thoroughly checked and has been found to be valid, the routine calls in the proper segment of the Job Description List so that the new entry can be merged into the List, using the Job Name as the primary sort key. The entered Job Description is printed out for manual corroboration and for use as an audit trail.

The deletion of a Job Description from the Job Description List is a relatively simple task because no other portion of the system is functionally dependent upon it. Only a Job Run Request, as it is being introduced to the system, is dependent on a Job Description, and the Job Run Request is rejected immediately if the proper Job Description is not in the system.

Consequently, the Job Entry Request Routine can delete a Job Description by retrieving it from the Job Description List and restoring the List without the Job Description. The name of the job must be deleted from each Program Description that is referred to by the job.