The Data Management component of the new IBM 8100 Distributed Processing Programming Executive (DPPX) provides for the storage and retrieval of data on disk and tape. Its objectives are to support a broad range of functions and be easy to use, be easily extendible, and entail minimal cost for the user. The Data Management component is designed to meet those objectives by means of a layered structure, an improved concept of device independence, and the use of catalogs.

Data Management for the Distributed Processing Programming Executive (DPPX)

by A. K. Fitzgerald and B. F. Goodrich

For the Distributed Processing Programming Executive (DPPX) of the IBM 8100 Information Processing System,[1] the Data Management component was designed and developed with the understanding that a variety of functions must be supported currently, and that patterns of usage will continue to evolve. The general objectives and structure of DPPX are described in the accompanying paper by S. C. Kiely.[2] The Data Management component provides for the storage and retrieval of named data and the creation and maintenance of data aggregates. The aggregates definable in DPPX include *records*, *data sets*, and *catalogs*. As used in this context, a record is a collection of related data or words, treated as a unit. A data set (similar to a *file* for readers more familiar with that term) is a named collection of records. A catalog is both a collection of the names of data sets and a collection of those data sets.

data management objectives

One of the objectives of DPPX Data Management is that it support a broad range of functions for its users. User data accesses can be expressed in COBOL or FORTRAN statements, in low-level assembler macro instructions, or in DPPX commands. These accesses can range from short, interactive operations to massive batch operations, and they can include single-site and network operations.

Short, interactive operations include DPPX command processing and the transaction processing provided by the Data Base and Transaction Management System (DTMS).[3] Users can access logical records sequentially, by record number, or by record key, or they can access a group of logical records, an entire data set, or a collection of data sets (an entire catalog).

Accesses by a single user often involve multiple data sets. Multiple individual users often have to access the same data concurrently, whether for inquiry or updating. Data can be shared between interactive and batch operations.

A second objective is that DPPX Data Management be easy to use; that is, new data sets should be easy to create, and it should be easy for new users to access the data sets. User access to data sets should be easy to control, and the programming interface for data set access should be relatively simple in order to facilitate the development of new DPPX application programs.

A third objective is that it should be relatively easy to add new function to, or modify, existing DPPX Data Management support in a compatible manner; that is, DPPX Data Management should be easily extendible. Extendibility is important because, with the continuing evolution of data usage patterns and storage hardware, DPPX Data Management will be subjected to additional functional requirements. The introduction of new data organizations or a new disk or tape storage device should result in the addition of new capability at the appropriate level within DPPX Data Management; it should not result in a top-to-bottom change of existing Data Management or a disruptive change to the user program interface.

Lastly, DPPX Data Management should be provided at minimal cost to the user.

comparison with prior systems

Because DPPX is used in the distributed systems environment, various requirements become important for DPPX Data Management that are not necessarily as important for prior systems. Already mentioned is increased emphasis on ease of installation and use because distributed systems are not expected to be staffed by computer professionals; distributed systems operate where the user's main tasks pertain to his business, not to his attending to a computer.

The ease of use necessary for effective operation of distributed systems suggests simple, flexible, nonredundant services. The strategy of prior systems with much specialization gave rise, for example, to a variety of access methods (SAM, DAM, PAM, ISAM, VSAM, and others) which by-and-large could not operate on the same data. This restriction made it difficult for users to change

their data usage as their needs changed. DPPX separates the concepts of data organization and data interpretation from the data recording format (when a direct access storage device is used), so the access pattern, whether direct, sequential, or keyed, can vary across the same data set.

The ease of installation necessary for distributed systems includes ease of installing applications. Thus programs should be able to migrate from batch to interactive operations, and from command to transaction environments. In DPPX, the user program interface is identical for batch, interactive, and transaction programs; there is no need for system generation, data base generation, or special program generation.

In older structures, data recovery provided by data base management subsystems is available only for the data organization handled by the subsystems. Conversely, such a data organization is processable only through the data base management subsystem. Again, DPPX with DTMS has demonstrated the ability to separate recovery from data organization. Thus user programs can build and process an indexed data set by using the DPPX Base.[1] Then, with no change to the user program or to the data set itself, the user can, with two commands, use the Data Base Manager of DTMS for full recovery. Similarly, data bases prepared by using DTMS can be processed through the DPPX Base, for example by various utility programs, given the proper authority.

meeting the objectives

The key design feature that enables DPPX Data Management to meet its objectives is a layered structure, which avoids implementation of redundant function (and redundant user interfaces) while allowing several different user functions to be supported. The inherent simplicity of this structure facilitates the designing, developing, and testing of user programs. The layered structure forces localization of the implementation of any function, so it is relatively easy to add or replace functions.

A second design feature of DPPX Data Management is improved device independence, which results in a simple programming interface to data set access and offers the user much flexibility in retrieving data from or transmitting data to various resources such as disk data sets and terminals.

A third design feature is the use of catalogs to facilitate a broad range of user interfaces. Catalogs provide a convenient and controllable means of grouping data sets.

This paper discusses these three features of DPPX Data Management and their relationship to the design objectives. The layered structure, because of its central importance to the entire DPPX structure, is discussed in greater detail than the other two.

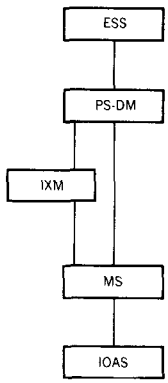
Layer description

The layered structure yields the design benefits predicted by Dijkstra,[4] such as greater certainty when testing because of dependencies on lower layers. The variable layering structure permits Data Management to offer much flexibility to the user, integrity to the data, and extensibility to the system, while granting the user improved physical data independence—improved beyond earlier generations of operating systems because of the ease of adding or replacing layers.

The layers of Data Management have well specified duties, and the boundaries between layers are opaque. The basic relationships among the Data Management layers and other DPPX input/output components are described in the accompanying paper by Albrecht and Thomason.[5]

From one vantage point (see *Media Services*, below), all data sets in DPPX have the same organization; only the usage varies. That is to say, a data set that appears to the user to be sequential has the same underlying structure as a data set that appears to be an index. Varying degrees of interpretation are given to data sets (or rather to the use of data sets) by a layered structure within Data Management. This structure permits higher layers to take advantage of the services of lower layers in a systematic way. This paper describes enough about the layers of Data Management to demonstrate how important the layered structure is in meeting the design objectives.

Figure 1 Usual data management layered structure



As the result of an OPEN FILE statement in a COBOL program,[6] a user's program is connected, through the layers of Data Management, to the data set that corresponds to the file. Each READ (or WRITE or REWRITE) statement by the program usually becomes a RECEIVE (or SEND) request at the interface to Data Management. The External Support Services (ESS) layer (see Figure 1) validates each request by verifying that the user has been properly connected to the data set. Then it embodies the request in an internal control block, which is passed as a parameter list across the layers. Thus, rather than using the operands of the user's request directly, Data Management layers use the internal control block. This embodiment frees the user's program to possibly proceed in parallel with Data Management processing, isolates the user-Data Management interface from interface changes within Data Management, and permits new Data Management layers to be substituted or added.

The Presentation Services for Data Management (PS-DM) layer operates on *logical records* for its users. (*Logical record* is a synonym for *record*.) Multiple types of Presentation Services are built into Data Management, as described below. The Index Man-

ager (IXM) layer operates on *keys* and *indices* (that is, sets of ordered keys). A key usually is a data field within a record and is used to identify that record. If logical records or keys are inappropriate, one or both of these layers can be omitted when the connection is established.

It is important to observe that the layer is omitted, not bypassed. There is no residue of a "missing" layer. This is important because architecturally there is no specific number of layers. Unknown layers can be added to perform some new function.

The software appropriate to each layer is always re-enterable (internal locks are used as needed). Therefore the program modules are shared among all instances of the same layer where appropriate. The layers that are built during CONNECT processing, as the result of a COBOL OPEN statement, for example, are control blocks that represent the use of a service or resource.

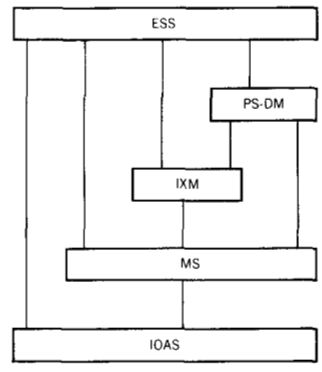
The Media Services (MS) layer controls *logical blocks* within a data set. A logical block is the smallest unit within a data set that is transferred to or from a storage device. As discussed below, Media Services enables its invoker to view a data set as a set of logical blocks while concealing the storage device.

The Input/Output Attachment Services (IOAS) layer causes data to be transmitted between main storage and secondary storage, such as disk.

In this design, each layer adds a greater degree of interpretation to the data than do lower layers. Thus PS-DM handles records, whereas IOAS handles only transmitted bytes.

A user who desires more exact control can connect for a level of service below that described in Figure 1. Such a connection is illustrated in Figure 2. The ESS layer always handles validation, but the PS-DM, IXM, and MS layers can be omitted, as shown, although the program will lose physical data independence.

Figure 2 Variable connection capabilities



As a result of the structuring of Data Management function into a set of layers, redundant function implementation is avoided while the user is provided with a variety of services. This "building block" approach allows different user interfaces to be supported with less design, development, and testing cost, and it ensures extensibility, as in the later incorporation of new services in a compatible manner.

In the following paragraphs, the various layers of DPPX Data Management are discussed more completely.

Input/Output Attachment Services

It is the duty of Input/Output Attachment Services (IOAS) to attend to the physical characteristics not directly handled by the device itself and to make storage devices within a device class appear similar. Thus IOAS is concerned with device addressing and device command syntax and semantics. It operates upon devices, volumes, and physical blocks, but not upon data sets. The unit of data transfer between IOAS and its invoker is a set of physical blocks.

Disks and diskettes that attach to the IBM 8100 provide for storage and retrieval of fixed-length physical blocks. Physical block size is architecturally an exact divisor of 256 bytes. Data sets, in DPPX, contain fixed-length logical blocks, which are always an exact multiple of 256 bytes. The size is determined by the user when the data set is created; however, the maximum is 4096 bytes. Thus a logical block always contains an integral number of physical blocks.

Tapes that attach to the IBM 8100 provide for storage and retrieval of sequential physical blocks. For tape data sets, in DPPX, logical blocks can be any size from 18 to 4096 bytes. The maximum size is specified when a tape data set is defined. The size of a physical block accessed by IOAS is the same as the logical block size.

Media Services

Early in the design of DPPX Data Management, it became clear that user application programs should be able to access the logical records or the logical blocks of a data set. Also, it is necessary for the logical record access service routines to access logical blocks when processing user application program requests because logical records, as implied earlier, are mapped onto logical blocks. Thus a design decision was made to develop a layer that would access logical blocks, and that layer was to be used for both logical block access purposes. Its use was extended to other purposes, as in the supporting structure for indices. As a result, there existed a requirement for a layer that would provide access to logical blocks of a data set without requiring information as to type of invoker. That layer is called Media Services (MS).

MS provides for user access to logical blocks of a data set on a disk or diskette volume or on tape volumes. MS does not interpret the contents of a logical block, so it is independent of data set organization.

A particular data set resides on portions of one disk or diskette or one or more tape volumes. Each portion is called a data set *extent*. Each extent is made up of fixed-length logical blocks. An extent comprises a consecutive set of logical block numbers assigned to a consecutive set of physical block numbers. The breakpoints between extents are unknown to the invoker of MS. Beyond specifying, when defining a disk or diskette data set, an

appropriate number of logical blocks (or logical records) to be contained in an extent, the user is removed from extent creation, maintenance, and manipulation considerations.

For a multiple-volume tape data set, the transition between volumes is handled by MS. And for a multiple-data-set tape volume, positioning to a data set also is handled by MS.

An understanding of the layered structure of Data Management and how it satisfies the design objectives requires an understanding of the internal structure of MS. Establishing an instance of the MS structure for accessing a particular data set is a three-step process:

- Activation of MS support for a particular disk, diskette, or tape device.
- Mounting the tape or diskette volume on which the data set resides.
- Establishing a connection between the user's application program and the data set.

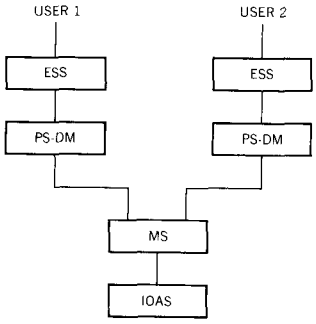
MS support for a particular disk, diskette, or tape device is established by means of an ACTIVATE command for the corresponding device. As a result of the activation sequence, an MS-to-IOAS connection (port) is secured, and IOAS presents to MS an interface to the particular disk, diskette, or tape device. A Media Services control block is constructed which contains information about the device, such as the port to IOAS, and which will represent the disk, diskette, or tape volume.

For either diskette or tape, the system operator is advised by Data Management to mount the required tape or diskette volume, if it is not already mounted, when a data set on the volume is to be accessed. The identification of the volume is found in a catalog, in a profile for the data set. As a result of the activation and mounting procedures, the MS control block pertaining to the device includes information that represents the volume.

A connection between the user's application program and the data set is established by issuing a CONNECT macro or command[7,8] or the source language equivalent (for example, OPEN in COBOL). During this process, MS uses a collection of information about the data set, called a data set profile, to set up the control block structure necessary to process logical block accesses to a data set. A data set profile is created whenever a data set is declared to DPPX by the DEFINE.DATASET command.

The connection processing constructs a set of MS control blocks that contain information about the data set. These control blocks are attached to the MS control block that represents the disk or

Figure 3 Two users sharing one data set



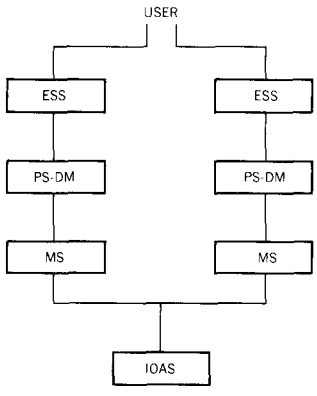
tape volume. In addition, a connection (port) to this MS representation of the data set is established. The subsequent invoker of this port (for example, a PS-DM layer, or ESS in the case of a user accessing logical blocks) can then process logical blocks of the data set.

MS is device-class dependent; that is, different MS processing routines and control blocks are used for disk data set access than for tape data set access. Because of the different capabilities of tape and disk, the invoker of MS (PS-DM, for example) must be given information about the device class, but only if the user wishes to take advantage of those differences (primarily the random addressing capability of disk storage). However, MS is independent of the particular device type; it requires no information about whether the device on which a particular data set resides is an externally attached diskette or physically within, say, an IBM 8130. When MS support for a tape or disk volume is established, the capacity of the volume is part of the information given by IOAS to its invoker, which usually is MS.

Presentation Services

Just as each instance of Media Services represents a data set, each instance of Presentation Services for Data Management (PS-DM) represents a use of logical records of a data set. PS-DM uses an instance of Media Services to access logical blocks so that it can operate on logical records of a data set for the PS-DM user. If multiple concurrent users attempt to share a data set, then multiple instances of PS-DM, one for each user, will connect to the same instance of Media Services, because Media Services represents the data set (see Figure 3). Record locking, when necessary, is exercised by PS-DM.

Figure 4 One user using two data sets on the same volume



Each instance of PS-DM represents the use of logical records of a single data set. Therefore, if one user is connected to two (or more) data sets, the connection process creates two (or more) PS-DM layers, each using a different Media Services instance (see Figure 4). The number of instances of IOAS used by the Media Services layers is determined by whether the data sets are on the same volume or on different volumes.

Logical records are processed by PS-DM. For each data set, the logical records have a fixed length, according to the user's specifications, with a maximum size of 4K bytes. The number of logical records per logical block is constant for each data set and is always greater than or equal to one. If that constant is not an integer on disk, the residue of the logical block is padded. It must be an integer on tape for compatibility with prior systems.

The characteristics of a data set that are important to PS-DM are housed in the data set profile. This is the same profile that is discussed above under *Media Services*.

The assignment of logical records to logical blocks is handled by PS-DM, thus removing from the user (rather than from the definer of a data set) all logical block considerations. That is, the user can view a data set as a set of logical records only. The logical records of a data set are assigned relative record numbers (RRN's) starting with 1.

In DPPX Base Data Management there are two kinds of PS-DM layers, the relative sequential access method and the Indexed Record Processor. The relative sequential access method version, usually called just PS, is appropriate when the logical records are accessed physically-sequentially or directly by reference to an RRN. Physically-sequential access is access by consecutive RRN's, which in turn means access by consecutively numbered logical blocks independently of their mapping onto physical blocks and extents. The Indexed Record Processor (IRP) operates on records within a data set that has a companion index.

A third kind of PS-DM layer, the Data Base Manager, which is provided by DTMS, processes indexed linear data bases and offers full recovery. Both the Indexed Record Processor and the Data Base Manager use the same intermediate layer, the Index Manager (IXM), for index operations.

sequential or direct access

For sequential or direct access to a data set, the connection process constructs control blocks that ensure the use of the PS service routines and of the Media Services port that represents the corresponding data set, as shown in Figures 3 and 4. For strictly sequential access, as for reading consecutively all (or some) of the records of a data set or inserting new records at the end, the user need not be aware of the relative record numbers. During sequential reading, deleted records are automatically skipped. That is, PS maintains the incremented value of RRN and can detect whether a record exists for a given RRN. It continuously increments the RRN until it finds the next existing logical record or an end-of-file condition.

Because user programs may have no information about the RRN's while sequentially reading or writing, the initial loading of a data set and the appending of new records can appear the same to the user programs. Of course, information is provided that allows a user program to be sensitive to relative record numbers and to whether the data set is empty at the outset.

The same data set that is processed sequentially can be processed directly by user programs. The degree of concurrency is contingent on user specifications and system limits that prevent deadlock. For direct processing, the user specifies the logical record to be read or written by its RRN. The PS layer uses the RRN to access the appropriate logical block and provides the blocking and deblocking of logical records to or from logical blocks.

Because data sets viewed as sets of logical records can be accessed either directly by relative record numbers or sequentially, they are called *relative sequential* data sets.

It should be observed that an instance of PS represents a sequential or direct usage of the logical records of a data set, and that the precision of the definition of the functions of a layer has led to much of the flexibility of the system. So, for example, a single user can connect for multiple uses of the same data set within the ground rules of sharing alluded to earlier and in other publications.[9] Thus a single user may be inserting at the end while reading from the beginning of the same data set.

indexed access

In addition to being able to process the same data set sequentially or directly, the user may decide that a field within the logical records is a useful key. The user can then request the system to build an index[7] for that relative sequential data set using that key field. The user can then process that same data set by using the index. This type of access involves one of the other two PS-DM layers, the Indexed Record Processor or the Data Base Manager, as well as the Index Manager.

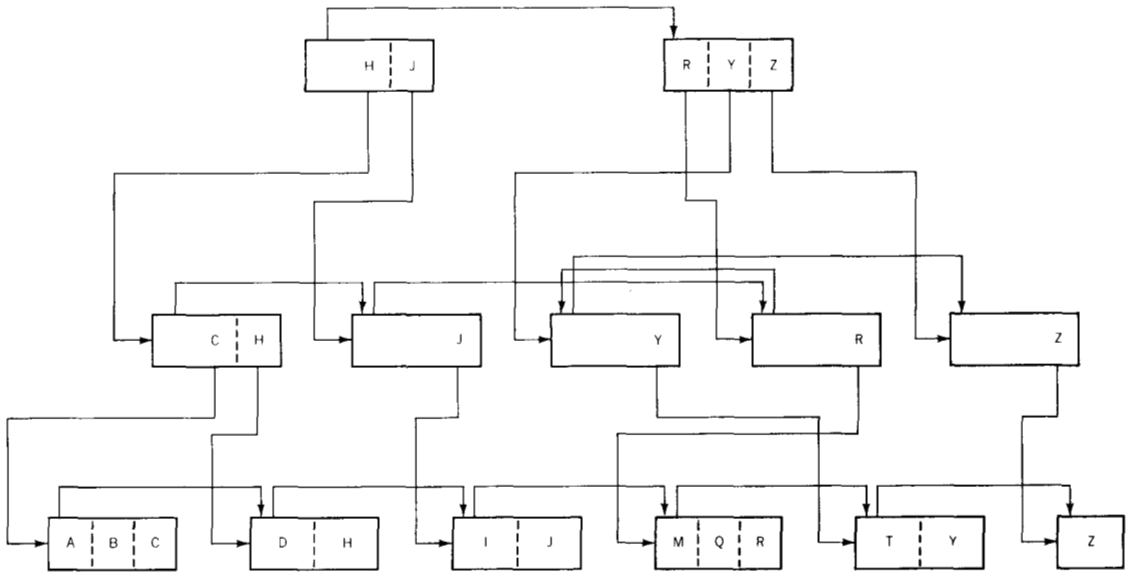
DPPX is designed with emphasis on interactive operations and for use in a network of homogeneous or heterogeneous computers and terminals. Thus the design anticipates that typical access to logical records on disk will often be random. To aid the systematic yet random access to data, DPPX provides for indexed access to records and an index structure.

For many reasons, including consistency of definition and convenience of use, it is important that a record identifier be invariant with respect to time and place. The invariant record identifier is a key, usually imbedded in the record. When logical records are copied from one data set to another, on the same machine or a different one, the keys remain the invariant identifier of the record. The ordering of the keys of a data set constitutes an index.

In particular, an index data set in DPPX is a well ordered set of keys, each with an associated pointer. An Index Manager, implemented as a layer for Data Management, maintains the order of the keys within an index data set. Each index entry of an index data set contains a control byte, a key, and a pointer. The index entries are assembled into an index block, which is identical to a logical block. Thus the Index Manager gains all the benefits of any other user of Media Services regarding the processing of logical blocks of a data set.

The index entries are prepared so that they form an inverted tree structure of index blocks, thereby ensuring an efficient search whether a read access is random or sequential with respect to the

Figure 5 An example of an index data set, a strict inverted tree with two roots



order of the keys. The pointer field of each index entry of an index block that is not at the leaves of the tree (for example, a root block) identifies by logical block number the index block on the next level that corresponds to that entry. Therefore the search can be repeated to find the index entry on a leaf of the tree that corresponds to the key of the search. The index is in the general form of a B^+Tree as described by Comer.[10] See Figure 5.

In addition, the index blocks on a level are laced together, thereby enabling sequential index entries within a level to be accessed. This technique not only optimizes GET NEXT requests, it enables operations to continue in spite of hardware or software malfunctions that prevent the maintenance of index entries on a level closer to the root during an insertion of a new key.

The pointer in each index entry on a leaf of the index tree identifies the logical record, by its relative record number, in the associated relative sequential data set, which is called its *target* data set.

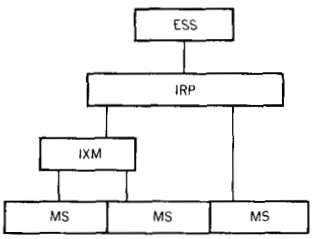
From the point of view of the Index Manager, the interpretation placed on the pointer of the index entries at the leaves is of no consequence. Again, employing the design principle of DPPX, each layer performs a well defined service and no more. Therefore, the Index Manager does not get involved in interpreting the significance of the argument associated with a key. Its duty is to maintain the strict tree structure of the index, to insert, update,

and delete index entries at the leaves, and to fetch an index entry either by key (key $= x$, or key $>= x$) or next sequentially from the last request by the same user of the index.

This strict adherence by the Index Manager to its duties has led to an easy extension of index operations to permit the accessing of stand-alone index data sets (those without an associated target data set). In this case, the index entries on the bottom (the leaves) contain data fields of a fixed length determined when the data set is defined.

Thus a user can define a data set as a relative sequential data set which he intends to use or to share with other users in any combination for logical block processing, sequential or direct logical record processing, or through an associated index. Also, a user can define a data set as an index data set with the data in the index. There are, of course, enforced ground rules governing modification of data sets to ensure data integrity.[8,9]

Figure 6 Connection for an indexed data set with two associated index data sets



Because it may be important to access the records of a target data set by more than one key—that is, to consider the records as having more than one sequence—it is possible to define up to eight index data sets across the same target data set. Each index corresponds to a different key field within the logical records. So for *index-target set* operations, at least two data sets are treated more or less as a unit, namely the target data set and at least one index data set. The Indexed Record Processor, IRP, uses the Index Manager, which in turn uses Media Services, for operations on the index data set. IRP also uses Media Services for accessing logical blocks of the target data set. Thus IRP provides services for the indexed access of logical records of an indexed data set.

All the index data sets that pertain to the same target data set form a set of indices. It is the duty of the Index Manager to maintain the integrity of the set of indices. Information passed from IRP to IXM on every update, insertion, and deletion makes it possible for IXM to maintain the set. This is done in such a manner that IXM is not given information about the interpretation of the pointer field of index entries at the leaves. For example, on an insertion, a pointer value and all the keys within the record being inserted come across the interface to IXM. All indices in the set have the appropriate key inserted with the specified pointer.

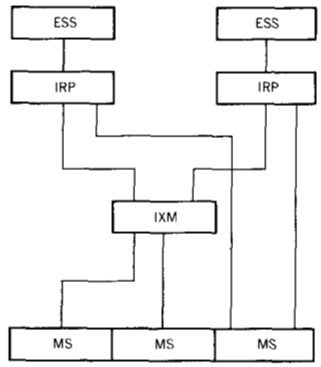
Considering first an unshared data set, during the connection process for the use of logical records of a target data set with two indices, the following layer structure is established (see Figure 6):

- Control blocks that represent the three data sets are prepared for the Media Services layer (that is, three instances of Media Services are constructed).

- Control blocks that represent the two index data sets and their usage are prepared for the IXM layer (one IXM instance for the set of indices).
- Control blocks that represent the use, via an index, of logical records of a target data set are prepared for the IRP layer (one IRP instance).

When there is sharing, there is another instance of IRP with an additional port from the same instances of both the Index Manager and Media Services (see Figure 7).

Figure 7 Two connections to the same indexed data set and index data sets



The extensibility made possible by rigid interfaces and exacting definitions of the duties of each layer is exemplified by the fact that IRP is replaced by the Data Base Manager of DTMS with no changes required in ESS, IXM, MS, or the connection process.

Data Base Manager

The Data Base Manager (DBM) portion of DTMS is designed for strict compatibility with IRP. User programs that operate on index-target sets by means of IRP can use DBM with no change. The converse is not necessarily true because additional data base services may be used. Also, an indexed data set and its associated indices can become an indexed linear data base, and vice versa, with no change to the data sets. Some additional control attributes must be declared by a user when defining a data base to benefit from the recovery facilities provided by DTMS.

The DBM operates as a "Presentation Services" layer even though it is licensed and installed separately after the DPPX Base. The installation process for DTMS adds modules to the program library and entries to system tables used by the connection process. Thereafter, operationally, DBM appears as another access method. It uses IXM and MS much the same way IRP uses them. DBM is invoked by ESS identically to any Data Management layer.

In addition to serving user program requests for sequential or keyed access to logical records of a data base—for retrieval, insertion, updating, and deletion—DBM provides improved deadlock detection and prevention compared with the DPPX Base, it provides the ability to back out changes to a data base if the work unit is not completed satisfactorily, and it provides the ability to recreate a data base from a prior checkpoint by re-entering intervening modifications from an audit file.

The extension of Data Management to Data Base Management is made possible by the compatibility of user interfaces and data structure on the part of IRP and DBM, and by the opacity of component interfaces within DPPX.

layer summary

The layered structure of DPPX Data Management provides a broad range of functions for users. A user can access physical blocks,

logical blocks, stand-alone index entries, or logical records. Logical records can be accessed either sequentially or directly by a relative record number or a key. This structure makes DPPX Data Management easy to use because data sets can be installed in a variety of ways.

The layered structure also makes DPPX Data Management easily extendible. It provides a variety of logical record interfaces, for example, all of which use the same Media Services support, and it enables various functions such as extent manipulation of Media Services to be supported for logical block and logical record accesses.

Lastly, the layered design allowed DPPX Data Management to be developed and tested at minimal cost, since support for functions is implemented only once in the appropriate layer.

Device independence

DPPX Data Management insulates programs and data sets from definitional changes in each other. Typically, a user's application program is connected to a data set by means of a CONNECT macro,[8] which may have been compiled from a COBOL OPEN FILE statement. Because program modules often cannot be permanently bound to a particular data set, commands make the data set name indirectly available to the program and to Data Management at execution time. That is, the set of commands that includes the invocation of the program can establish the conditions for the program, such as its data sets. The conditions are established primarily by an ASSOCIATE command[7] which relates the data set name to a synonym used within the program. Thus ASSOCIATE, which easily can vary from one execution to another, and CONNECT, which usually is stable within a program module, together relate an instance of a program's execution to a data set.

This indirect relationship between a program module and a data set plays a key role in providing a simple programming interface to data set access. (Resources other than data sets also are indirectly related to the using program.) This relationship allows one to write a generalized program, which can operate on any data set meaningful to the program, simply by changing the name of the data set on the ASSOCIATE command. As shown below, for example, an accounting program (here called ACNTNG) can operate each month on the data set that contains the month's ledger.

In February:
ASSOCIATE Synonym-1 ResourceName = FebruaryLedger
CALL.PGM ACNTNG

And in March:
ASSOCIATE Synonym-1 ResourceName = MarchLedger
CALL.PGM ACNTNG

General utilities of DPPX, such as the COPY.DATA command processor, also take advantage of the indirect relationship between a data set and the using program. The COPY.DATA utility gains even more flexibility by, in effect, executing the ASSOCIATE command. It gains information to do so from its operands or from a catalog.

In addition to making the data set name a variable, the ASSOCIATE command enables the resource type to be a variable, thus providing another degree of flexibility. A user's program can, in separate executions, operate on a terminal, a data set, or a data base, thereby, as mentioned above, implying a much higher degree of recoverability.

Another aspect of achieving a simple user programming interface, related to the flexibility of associations, pertains to the high degree of device independence for users and user programs that is effected by Data Management. For user programs that access sequential records within a data set sequentially, the data set can reside on disk, diskette, or tape. The user interface for such access extends naturally to records not in data sets—for example, records transmitted to or from terminals or other programs. Data sets whose records might be accessed nonsequentially can be on disk or diskette. Although such device independence, while important, is not novel, the interchangeability of data sets and terminals *is* novel. It is also practical; for example, in testing a program, input text from a terminal can be simulated by a data set.

For accessing data sets on disk, additional independence is afforded to programs; in particular, the access mode is independent of data set organization, as explained earlier. Thus sequential and direct processing can apply to all data sets—whether created by a sequential load or a sparse random operation, or indexed by keys. Explicit indexing operations do require the existence of an index; however, some indexing operations can be implied. For example, if a COBOL program executes the statement

READ WITH KEY = x

then DPPX can fetch the record with key x if the data set is an indexed data set. This operation is invalid on a data set without an index. On the other hand, if a data set is accessed sequentially, it is irrelevant to the program whether it is indexed or not. If it is indexed, then the records are accessed in the order determined by the index. If the data set has multiple indices (more than one key per record), then the index of reference can be decided by use of the ASSOCIATE command. Thus, greater separation is achieved between programs and data set organization.

In summary, device independence provides:

- A broad range of services for a user program by enabling a program without change to use various resources.
- Ease of use for user programming and application development and maintenance.
- Extensibility of customer configurations by being able to substitute new resources freely.
- A means of assuring minimal cost for user program development and maintenance.

Catalogs

Catalogs aid in the installing of programs and data sets by providing a convenient means of control for a group of related users. The importance of catalogs is derived from the fact that they house data sets and their descriptors. Catalogs are themselves special data sets, operated upon in an almost recursive manner. A catalog from one point of view is a data set, while from another it is a collection of data sets and an allocator of disk or diskette space. The fact that a catalog is both a data set and a collection of data sets assures extensibility in the future development of catalogs.

Unless a data set is declared to DPPX Data Management, it does not exist; once defined, the data set exists within a catalog.

Initially, by the design of Data Management, there is a *master catalog*, which exists on the system residency disk volume. The master catalog controls the space on that disk volume and controls all other disk volumes declared to DPPX Data Management. All data sets and catalogs with a simple name are listed in the master catalog.

There are two other types of catalogs: user and volume. A *user catalog* controls a portion of a particular disk or diskette volume. A *volume catalog* controls an entire volume. Volume catalogs are useful for the portability of data sets. For example, a diskette that is defined as a volume catalog and that contains data sets and user catalogs can be transported to another DPPX installation and be accessed there merely by entering the definition of the volume catalog. Its constituent data sets become immediately available.

User catalogs are comparable to libraries on previous systems. By defining data sets within a catalog, the user can create a collection of data sets, restrict access to the collection, and in general maintain tighter control of those data sets. Also, user catalogs enable pre-allocation of space to effect physical proximity of related data sets. In some instances, physical proximity is desir-

able to enhance performance. In addition, user catalogs are used by other components of DPPX, such as the Command Facility,[1] and they contribute to the easy assembling of data sets with similar attributes. This feature is especially useful for catalogs that are treated as data sets—for example, the DPPX standard program module library.

A catalog is a single data set composed of three parts:

- An index of data set names (the same index structure as that used by the Index Manager).
- A series of profiles.
- The data sets and catalogs defined within this catalog.

Catalog Management

Catalog Management functions provide for data set and catalog creation and deletion, including alteration and displaying of the characteristics of an existing data set or catalog. They also provide for disk space management; that is, creating and deleting disk space, and allocating and deallocating disk space to and from data sets and catalogs.

In addition, Catalog Management provides a naming convention for data sets and catalogs with qualified names, enabling two levels of nesting to occur for data sets and catalogs. Specifically, a user catalog (B) might be defined within a volume catalog (A) with the qualified name A.B. In addition, a data set (C) might be defined within a user or volume catalog (A.B or X), and its name would be qualified A.B.C or X.C. Data sets X.C and A.B.C would be entirely different.

Finally, Catalog Management provides for detecting and recovering damaged catalogs and data sets.

In addition to using these Catalog Management functions, a user can access a catalog as a single data set because of the simplicity and consistency of the data access interface. However, such accesses are not allowed to violate the integrity of the catalog. This kind of catalog access is especially advantageous to the performance of a user program that accesses similar data sets in a read-only manner.

Thus, the design of DPPX catalogs offers a range of services including portability, data space control, and a data set naming convention. Catalogs contribute to ease of use because they provide for nested management of the constituent data sets, thereby allowing a user to control his data in a variety of ways. Also, because of the naming convention and portability assistance, new applications can be easily installed or transported to new 8100 systems. Because of the recursive implementation for processing data sets and catalogs, redundant facilities are avoided.

Summary

DPPX Data Management, together with DTMS Data Base Management, is designed using a layered structure in which each layer implements its functions strictly according to specific interfaces at its upper and lower boundaries to provide a consistent set of services. The opacity of the interfaces and the strictly defined functions provide a great degree of flexibility in designing new services. The Data Management design, together with other parts of DPPX, improves the concept of device independence. As a result, terminal operations and data sets can be processed by the same program without change. Data Management uses and provides catalogs which aid in the administrative control of data sets and storage and also offer programming conveniences. Thus its design enables DPPX Data Management to support a broad range of functions at minimal cost. It is extensible, and it is easy to install and use.

CITED REFERENCES

1. *DPPX General Information Manual*, IBM Systems Library, order number GC27-0400, available through IBM branch offices.
2. S. C. Kiely, "An operating system for distributed processing—DPPX," *IBM Systems Journal* **18,** No. 4, 507–525 (1979, this issue).
3. *DPPX/DTMS General Information*, IBM Systems Library, order number GC26-3915, available through IBM branch offices.
4. E. W. Dijkstra, "The Structure of the 'THE'-Multiprogramming System," *Communications of the ACM* **11,** No. 5, 341–346 (1968).
5. H. R. Albrecht and L. C. Thomason, "I/O facilities of the Distributed Processing Programming Executive (DPPX)," *IBM Systems Journal* **18,** No. 4, 526–546 (1979, this issue).
6. *DPPX COBOL Language Reference*, IBM Systems Library, order number GC26-3923, available through IBM branch offices.
7. *DPPX Commands: General Use*, IBM Systems Library, order number SC27-0404, available through IBM branch offices.
8. *DPPX Macro Reference*, IBM Systems Library, order number SC27-0413, available through IBM branch offices.
9. *DPPX Guide to System Services*, IBM Systems Library, order number SC27-0405, available through IBM branch offices.
10. D. Comer, "The Ubiquitous B-Tree," *ACM Computing Surveys* **11,** No. 2 (June 1979).

The authors are located at the IBM System Communications Division laboratory, Neighborhood Road, Kingston, N.Y. 12401

Reprint Order No. G321-5109.