VAX/VMS Software
Information Management Handbook

VAX/VMS Software
Information Management Handbook

digital

Contents

Chapter 2 ▪ Case Studies in Success With The VAX Information Architecture

Chapter 3 ▪ The VAX Common Data Dictionary (CDD)

Chapter 4 ▪ VAX DATATRIEVE

Chapter 5 ▪ VAX DBMS Database Management System

Chapter 6 ▪ VAX Rdb/VMS

Chapter 7 ▪ VAX TDMS and VAX FMS

Chapter 8 ▪ VAX ACMS

Chapter 9 ▪ VAX DECgraph

Chapter 10 ▪ VAX DECslide

Chapter 11 ▪ VAX VTX

Preface

The book you are now reading is one in a set of three describing offerings from Digital Equipment Corporation in the area of VAX/VMS software. More specifically, this three-book set covers VMS System Software, the associated Languages and Tools, and Information Management solutions.

▪ The VAX/VMS Software Handbook Set

The three volumes of the VAX/VMS software handbook set are meant to give you, the reader, a substantial overview of the covered products' capabilities, features, and benefits. They attempt to go into enough depth so that readers at various levels of technical proficiency can all find the book useful when investigating VAX software solutions. Every reader is different. Each of you will want to read different portions of one or more of the books. Some of you may want to browse or skim for as little as 10 or 15 minutes — others among you may want to spend several days, and to continually refer back to certain chapters.

Note however that the books are not intended to serve as user documentation. They are not meant to have the same depth of technical information as are user documentation sets for the various products. Note also that it is not the intent of this 3-book set to discuss every software product that can run on VAX systems. Products covered in the set simply fall into one of three closely related categories — system software, languages and tools, and information management. Not only are there many more Digital software products available for VAX systems, but several hundred third-party products as well. For information on such products, refer to publications listed at the end of this Preface.

▪ The Information Management Handbook

This handbook is written for the person with anywhere from moderate to no expertise in using sophisticated information management software. It attempts to explain to the reader a number of the key issues surrounding information management solutions — what to look for in the capabilities of your software, what to avoid, what might be a good approach given certain existing conditions in your organization.

The book begins with an "Executive Summary" that lays a foundation for the remaining chapters by discussing information management in general, overall terms. It presents a few key concepts, touches on problems and challenges that may be facing your organization today, and suggests that Digital's VAX Information Architecture may be the best total solution for you.

Chapter Two offers four possible customer scenarios which have as their purpose to illustrate some ways to get the most out of VAX Information Architecture products. You will come to see, from reading this chapter and the remainder of the book, that the possibilities are nearly unlimited for putting successful combinations of these software products to work in your environment.

The nine remaining chapters present the individual products that make up the VAX Information Architecture, one at a time. Attention is given throughout, however, to the ways in which each product can interact with others in the set. In each of these chapters, the product is defined, its typical uses and users are described, key features and the resulting benefits are explained, and a closer look is taken at certain major aspects of the product.

Finally, a glossary of terms often used in conjunction with Digital's VAX Information Architecture is included for your reference and convenience.

▪ For Related Information

Recall that a great deal of related information will be found in the two other handbooks that make up this set, the Languages and Tools Handbook and the VMS System Software Handbook.

But to learn more about various aspects of other VAX software, hardware, or systems, refer to any of the following publications. They can be obtained through your local Digital Sales Office or Sales Representative.

- The VAX Software Source Book — Volumes 1 (Application Software) and 2 (System Software). These books describe over 1,600 software products for VAX systems, both Digital and non-Digital, "third-party" products.
- The ALL-IN-1 Handbook
- The Digital Dictionary
- VAX/VMS Internals and Data Structures
- Documentation Products Directory
- Digital Software Product Descriptions (SPDs)
- The VAX/VMS Version 4 Technical Summary
- The VAXcluster Software Technical Summary

Chapter 1 ▪ Executive Summary — What is Information Management?

Organizations today have to cope with ever-increasing quantities of information. Controlling inventory, tracking customer credit, filing reports with government regulatory agencies, and analyzing trends are all examples of managing information. But what exactly does "information management" mean? The next few pages provide an answer.

▪ What is Information Management?

Information management refers to the software capabilities that a computer system uses to handle and manage data. Traditionally, these software capabilities have fallen into three general categories: file management, data management, or database management. A particular system may provide any one or all of these three capabilities.

File Management

The most basic and most common information management software capability is *file management* (sometimes referred to as *record management*). At this level of capability, programmers write application programs that use the services of a file management system to access data in single, unrelated files. These files are usually stored on disk and organized in a sequential, relative (numbered cells), or indexed sequential (ISAM) format. The applications are written in traditional high-level programming languages such as COBOL and FORTRAN.

Digital's VAX RMS (Record Management Services) is an example of a file management system.

In most cases, a program using a file management system must precisely describe within its logic the data to be managed by the program, and where this data is stored. For the program to access data, the data must be stored in exactly the form and location described within the program. If the form of the data or how the data is stored changes, the data descriptions in the application program must exactly reflect these changes. In other words, the application program and the data it manages are strongly *dependent* on one another.

Typically, in order to access data with a data management application, the end user must have some knowledge of how the application is written and how the data is stored.

In file management systems, data security is provided by the operating system or by the application program.

File management systems are used extensively in the structured (or Data Processing) environment, as well as in the departmental and ad hoc environments.

Data Management

The term "data management system" is usually applied to a system, or combination of systems, that provides such user-oriented tools as: query languages; report writers; and data dictionaries for managing logical records.

The combination of the VAX RMS record management services and VAX DATATRIEVE (with the VAX Common Data Dictionary) provides data management on VAX systems.

Database Management

Database management is the most sophisticated level of information management. One of the most important features provided by a database management system is complete data independence, which allows data definitions to be removed from the application programs that use them and stored separately in a common area. The advantage of data independence is that if a change is made to a data definition used by one or more programs, no change need be made to the logic of any of the affected programs; it need be made only to the external data definition (and therefore only once).

Database management systems typically provide comprehensive security features that can be used to limit access to data, and data integrity features that can be used to ensure the accuracy and consistency of data and to recover from hardware failures. They also provide sophisticated techniques for modeling data structures and data relationships.

Database management systems were developed originally for the structured data processing environment, but they are being used more and more frequently by departmental and ad hoc users.

A particular DP shop may have information management systems that fall into one or more of these major categories. It is most likely to have a file management system, and a harried staff desperately trying to develop new applications while updating and maintaining the current array of applications.

It is far less likely to have a sophisticated database management system. And even less likely to have an integrated information management system that has the capability to meet all current information management needs and to be extended to meet all future needs. More about this level of capability — the fourth level — in a moment.

▪ Another Perspective — Data Processing vs. Information Management

The terms *data* and *information* are often used interchangeably, but the two words have different meanings. Data refers simply to facts that are not yet related to one another. Examples include a list of part numbers, the names and addresses of your customers, or the number of hours a particular employee worked during a particular week. Information, on the other hand, is data organized in such a way as to answer a question or facilitate some organizational need. Price lists, for example, contain useful information because they relate individual products to prices. Turning data into information is generally referred to as *data processing*.

Whether your organization is computerized or not, you process large quantities of data every day. Multiplying the price of an item by the quantity ordered is data processing, yielding important information — the total invoice price. Checking the name of a potential customer against a list of customers who have exceeded their credit limits is another process that produces useful information. Figure 1-1 illustrates this difference between data and information.

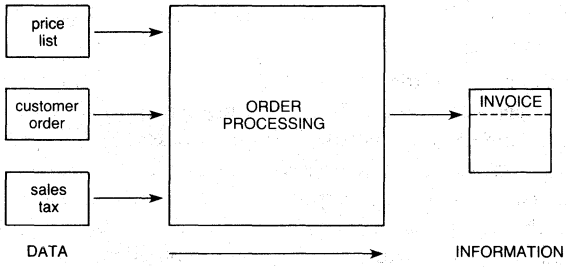


Figure 1-1 ▪ Processing Data To Yield Information

The ways in which you gather and store data and then transform that data into useful information can affect the success of your organization. Your data must be accurate, up-to-date, and readily available. Your data processing must be both efficient and reliable if it is to provide the information you need when you need it.

Data Processing

The goal of data processing, whether manual or automated, is to make useful information available. Typical processes include:

- Recording
- Searching
- Sorting
- Calculating
- Comparing
- Updating

Processing requests can come from any of several departments or locations, but data processing generally takes place in central departments where the accuracy of data can be controlled. Centralization relieves individual departments of the necessity of keeping track of changes made by other departments. So, for example, insurance agents forward changes in their clients' policies to a central office, where the changes are recorded and the new premiums calculated. When there is a claim against a policy, the adjuster gets the necessary policy information from the central office.

In organizations with automated processing systems, users bring their information requirements to a central data processing department staffed by professional programmers and analysts. This group designs and implements the application systems that meet user requirements. However, the process of translating needs into computer specifications and then maintaining applications once they are in place is expensive. As a result, central data processing is best suited to highly structured, high-volume applications, such as complete order entry or inventory systems. Special or *ad hoc* requests for information often remain unanswered for long periods of time because maintaining structured applications has a higher priority.

Information Management

Requests for information have been increasing steadily in recent years. The trend has led managers to seek solutions outside the data processing department. Instead of submitting all requests to a central group, department heads have begun to hire their own programmers to develop departmental applications.

Decentralizing data processing in this way increases overall efficiency, but at the expense of control. Traditional data processing requires that each application program describe the data and how it is used within the logic of the program. Programs and data, therefore, can be very dependent on one another, and if one changes slightly, the other must also. Redundancy and inconsistency result when different departments process data independently. Instead of accessing the central files, departmental programmers often duplicate data stored in the central files for their own applications. Subsequent updates to the central files are not included in the local copies, so that files become less and less reliable over time.

To enable organizations to maintain control over data processed locally by different departments, vendors have recently begun developing software products that keep the definition and management of data separate from application programs. With these information management products, individual departments no longer need to maintain their own data files, nor must data access originate in a central data processing department. Instead, processing can take place locally, while the software protects data against unauthorized access, redundancy, and inconsistency.

Information management makes it possible for users outside the data processing department to get needed information without concern for the details of its physical storage. Office workers and managers can examine data and format it as useful information. Different departmental data processing groups can simultaneously update the central files without interfering with one another. Programmers can update programs without having to redefine the files in which data is stored.

Correctly implemented, information management software can improve the overall efficiency of an organization's data processing. Relieved of the necessity of answering numerous *ad hoc* requests from users, the central data processing department can devote full attention to designing and maintaining structured applications. In other departments, information management tools let users develop their own applications to answer their own information needs.

▪ Managing Information: The Problems Facing the DP Manager

The typical DP shop today is having an increasingly difficult time meeting the requests of the organization for data processing. At best, it may be managing to cope; at worst, falling farther and farther behind.

There are several reasons:

- Manpower costs are rapidly increasing.
- Additional program development talent is growing more scarce.
- There is a push for more and more computerized processing.
- *Known* program development backlogs are typically three years. When the *hidden* backlog is also considered, (Many desired applications are never formally requested because of the already-huge *known* backlog.) total backlog tends to be approximately six to seven years.
- Data processing shops spend 60% to 70% of their budget on software maintenance.
- End users cannot get the information they need when they need it.
- Demand for system control (security, centralization) is increasing.

Some DP shops are attempting to relieve the pressure on their departments by distributing some of the data processing responsibilities out to the departments and end users making the requests. With the data processing responsibility, of course, goes the data. However, there is a built-in pitfall in doing this: reducing central control over data allows data redundancy. Data redundancy is multiple occurrences of the same data item, spread over one or more systems.

Data redundancy reduces efficiency by making it necessary to update several occurrences of a data item instead of just a single occurrence; and it can threaten the reliability of the organization's data, because there can be no guarantee that every occurrence of a particular data item spread across an organization will be kept up to date.

So what's the solution? How does the DP shop meet all the demands being placed on it while holding down program development costs and maintaining control over the reliability and consistency of the organization's data? With a set of coordinated, compatible information management tools that can solve a wide variety of problems for you.

▪ Information Management Tools

Many software tools are available for setting up efficient information management systems. These tools perform the following functions:

- Information resource management, providing central storage of data descriptions and record definitions
- Data access, allowing you to retrieve information easily
- Distributed processing, allowing you to process data stored on other computers remotely from your local system
- Reports and graphics generation, providing informative and attractive reports, graphs, and charts
- Terminal management, displaying familiar business forms on the terminal screen to make it easy to manipulate the data in your files
- Data and database management, controlling data shared by many users
- Application management, controlling large, complicated applications

Information Resource Management

Data in files is described by record definitions. Traditionally, these definitions have been included in the programs that process the data. The COBOL data division, for example, contains definitions for all of the data used in a COBOL program. *Information resource management* helps avoid a proliferation of files containing the same data defined differently. This approach makes data descriptions independent of program logic. The principal tool of information resource management is the *data dictionary*.

Data dictionaries define and describe all of the data items used by an organization. Instead of creating new files and record definitions as they perceive a need, programmers can use the data dictionary and the information resources that already exist.

Data dictionaries can be active or passive. Passive dictionaries simply store descriptions of data and generate listings of data definitions and available information resources. Active data dictionaries allow programs to extract data definitions as program source code. With an active data dictionary, you can create new applications, or modify old ones, without redefining data. Instead, you can include the dictionary definition automatically in your application regardless of language. Use of an active data dictionary increases efficiency and maintainability by reducing the number of program-specific definitions.

Data Access

To make it easy to retrieve data for processing, special *query languages* let you use everyday English words to perform tasks that used to require application programs. Important query language capabilities include:

- Searching files for information based on criteria you specify
- Sorting data
- Adding data
- Modifying data
- Deleting data
- Protecting data

For example, a query language lets you use a simple command to find the names of all your employees who earn between $25,000 and $30,000 a year:

```
PRINT EMPLOYEES WITH SALARY BETWEEN 25000 AND 30000
```

The query language finds all employees matching this criterion and displays information about them on your terminal screen.

Distributed Processing

Distributed processing gives you the ability to access data on remote computers as easily as you access data stored on your local computer. With distributed processing you can decentralize your data files without introducing redundancy or relinquishing control.

In a distributed processing environment, the data your department uses most frequently is stored locally. When a user on another computer needs to access your data, distributed processing software handles the physical data retrieval. From the user's point of view, there is no difference between local and remote processing. Distributed processing helps prevent data redundancy, because no new copies of the original data files are made.

Report and Graphics Generation

Report writers make it easy to retrieve data from central files or databases, to arrange and manipulate that data, and to produce informative and attractive reports.

For example, a simple summary report might involve printing the name and monthly revenues of each branch of a department store chain followed by the total monthly revenue. Producing this report with a traditional programming language requires the following steps:

- Create a variable TOTAL_REVENUE and set its value to zero.
- For each branch, add the monthly revenue to TOTAL_REVENUE, and then print the values of NAME and REVENUE in the report.
- After the last branch has been processed, print the value of TOTAL_REVENUE in the report.

With a report writer, you do not need to calculate the total explicitly. Instead, simple commands specifying what you want, not how to produce what you want, are all that is required:

```
PRINT NAME, REVENUE

AT BOTTOM PRINT TOTAL_REVENUE
```

Most report writers let you store report formats for future use, so that once you have defined a format, you can use it later to produce reports automatically. Whether you need a report that is used only once or one that the government requires you to file each month, a report writer allows you to produce the report quickly and easily.

Graphics generators are similar to report writers, but instead of producing reports, they present the data stored in your files as line and scatter graphs, bar charts, and pie charts. To allow you to create graphs without having to write programs, graphics generators usually offer a simple command syntax or a menu interface for graphic design.

Graphics are a dramatic way to change data into information; large quantities of information can be grasped at once, and trends quickly become apparent. Consider, for example, the difference between reading columns of figures and seeing a graph of those figures over time. Graphics programs can quickly produce sophisticated color displays of your data.

Terminal Management

In business, most data is gathered and stored on forms. Displaying such forms on a terminal screen provides a familiar and easy method for entering and retrieving data.

Many forms processors check values as the data is entered and accept a value only if it is of a specified type or within a specified range. For example, you can direct the form to accept a value for an employee code only if that value corresponds to one of the values listed with the form definition. Control of this kind leads immediately to fewer data entry errors.

Database Management Systems

During the 1970s, sophisticated *database management systems* (DBMS) emerged to provide greater control over data than that available with conventional file structures.

In general terms a database is, simply, stored data, but the term has a more specialized meaning in the context of database management systems. Like traditional files, DBMS files contain data and record definitions, but DBMS files also contain representations of the relationships among the data items and records. Instead of relying on traditional file access methods, DBMS software controls access to data and data definitions.

There are three basic database structures:

- Hierarchical
- Network, also called the CODASYL model because the COnference on DAta SYstems Languages has been active in developing network database specifications
- Relational

The *hierarchical* database organizes the relationships between record types as a tree structure. Related records are stored on the same branch of the hierarchy to facilitate efficient data retrieval. A disadvantage of the hierarchical structure is the lack of flexibility in navigating through the database: once you choose one of the branches, there is no way to get to the records on the other side of the branch without moving back up the tree to the junction of the required branch. At that point you can begin working down the other side of the tree.

In Figure 1-2, the hierarchical relationships are clear:

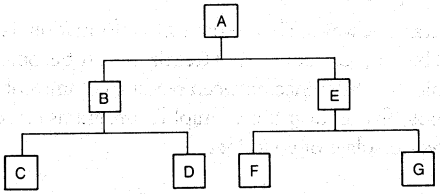


Figure 1-2 ▪ The Hierarchical Database Model

Records C and D are clearly related to Record B, which is, in turn, related to Record A. If you want to relate Record C to Record E, however, the hierarchical organization of the database requires you explicitly to link C to B, B to A, and A to E when you access these records in a program.

With the *network* or *CODASYL* model, any record can be related to any other record without the restrictions inherent in the hierarchical structure. Because records can participate in relationships, called *sets*, that are not limited to records hierarchically above and below, the network model provides flexibility in matching database structures to your data processing needs.

In Figure 1-3, the EMPLOYEE record participates in three set relationships:

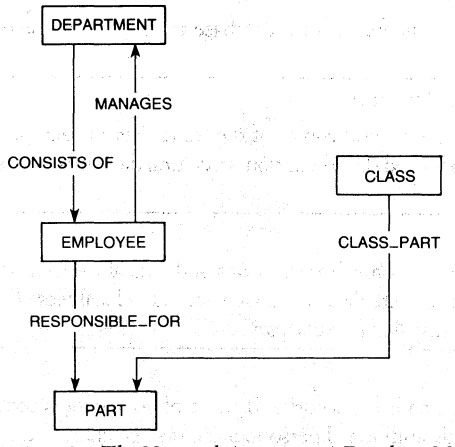


Figure 1-3 ▪ The Network (CODASYL) Database Model

Departments both consist of employees and are managed by them. In addition, employees are responsible for maintaining parts. By adding record types and sets in this way, you can use a network database to reflect the data relationships in your organization. The advantage of predefining relationships in network databases, especially databases containing large numbers of records, is processing efficiency.

The *relational* database model provides more flexibility than either the hierarchical or the network model because relationships do not exist as predefined structures. Instead, data is stored in tables, and relationships between two or more records are established by matching the values of *key* fields common to those records.

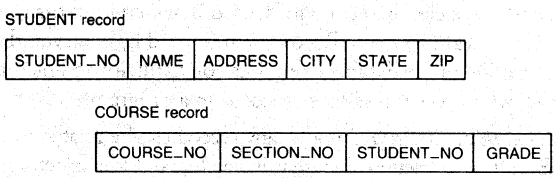


Figure 1-4 ▪ The Relational Database Model

In Figure 1-4, no relationships between students and classes are defined. Because the student number is common to both records, however, it is easy to associate a class number and grade with the name and address of the student who took that course and earned that grade.

In summary, implementation of a database management system can provide several benefits:

- Reduction in redundancy.

  Instead of storing several copies of the same data in each of several files, a DBMS stores data and data definitions in central files and controls the physical storage.

- Views.

  Individual users see only those portions of the database they need to do their work, and they update data as if it were stored in local files. These subsets of the database are called *logical views*.

- Security.

  DBMS software enforces security. If some of your data is sensitive, you can ensure that only authorized personnel can read or change it.

- Shared access.

  Because the DBMS controls access to the data, it is possible to control shared access to files. This means that many of your employees can update the database simultaneously without introducing errors. DBMS software is programmed to resolve any conflicts that might arise.

- Recovery from failure.

  As employees process database data, their transactions are recorded in a journal file. Therefore, the database can be restored to accuracy if hardware failures corrupt or destroy a day's database activity.

Database management systems provide these benefits because they perform many of the data handling and file control functions that must be performed by individual programs in a conventional file management system. Conversion to a DBMS from a conventional file system, however, can be expensive at first, in part because trained technical personnel are often needed to design and implement DBMS applications.

Application Management

As your information management system grows and becomes accessible to more and more employees, you need more control and more efficient processing of common data. Application management systems answer this need. Typically, application management systems let employees with little or no computer experience perform standardized data processing tasks by making selections from a menu displayed on a terminal screen.

Application management systems give you broad control over which menus each of your employees can see and use and over the tasks each employee can perform. These systems also provide facilities for logging the work users have done. Such data is necessary both for application security and for tuning application programs. With application management systems you gain the benefits of efficient data processing while minimizing the risk of granting broad access to your company data.

▪ Planning an Information Management System

Each of the tools previously described provides benefits, but no information problem has only one solution. Different tools, and groups of tools, solve different problems. For example, a query language and report writer might provide all of the information management needed by a company of 50 employees. On the other hand, a company that manufactures and distributes more than 1000 products should certainly investigate the benefits of a database management system. To make an intelligent choice, you must evaluate the information management products available in light of your particular needs.

The following chapters introduce Digital's family of information management products, the VAX Information Architecture. In addition to learning about the products, you will see how they work together in different combinations to answer different needs.

▪ Digital's Solution for Information Management

What the DP shop needs in today's data processing environment is a fourth level of information management capability — an information management "solution" that:

- Makes more efficient use of the current program development staff
- Improves productivity at all levels of the organization
- Provides a set of individual information management products that can be integrated or extended, as needed, to meet current and changing needs

Digital supplies such a solution with a family of information management products called the VAX Information Architecture.

▪ The VAX Information Architecture

The VAX Information Architecture consists of a family of information management products that can function alone or in concert with one another to meet almost any individual information management need. These products establish a modular framework that allows customers to choose a tailored information management solution (particular set of products) that satisfies their current needs, and to employ additional capabilities later on as the needs change — without impacting past, current, or future development investments.

The VAX Information Architecture addresses seven general functional categories of information management:

- Common Storage
- Data Access
- Data Management
- Distributed Processing
- Reports and Graphs
- Application Management
- Terminal Management

▪ How the VAX Information Architecture Products Work Together

The VAX Information Architecture is a modular resource whose component products can be used to customize a "solution" for almost any information management need or set of needs.

There are currently nine separate but interrelated products in the VAX Information Architecture family:

- VAX Common Data Dictionary (CDD)
- VAX DATATRIEVE (fourth-generation language and data management facility)
- VAX DBMS (CODASYL database management system)
- VAX Rdb/VMS (relational database management system)
- VAX FMS (Forms Management System)
- VAX TDMS (Terminal Data Management System)
- VAX ACMS (Application Control and Management System)
- VAX DECgraph (graphing facility)
- VAX DECslide (text and graphics presentation facility)
- VAX VTX (videotex presentation service)

The VAX Common Data Dictionary (CDD)

The VAX CDD is the hub of the VAX Information Architecture. It is the central repository for data definitions used by other VAX Information Architecture products and also, optionally, for data definitions used by application programs.

The CDD is required for using VAX DATATRIEVE, VAX DBMS, VAX TDMS, and the VAX ACMS. It is optional for use with VAX Rdb/VMS.

Storing data definitions in the CDD that would otherwise have to be defined within the logic of the apllication programs that use those definitions frees application developers from the need to define the data their applications use. In other words, the CDD makes it possible to remove data definition from the program development process. This not only saves development costs; it also fosters consistency, by requiring that only one copy of each data definition need exist, and it facilitates maintenance by requiring that only one copy of each data definition need be changed.

VAX DATATRIEVE

VAX DATATRIEVE is a data retrieval and reporting tool that you can use to access any data stored on a VAX/VMS system, including data stored in VAX RMS files and in VAX DBMS or VAX Rdb/VMS databases. Data can be accessed interactively or through application programs.

VAX DATATRIEVE also includes a graphics facility you can use to produce plots that are excellent for discovering trends and supporting decisions. If more sophisticated graphic displays are required, the DATATRIEVE/DECgraph interface allows you to take advantage of DECgraph's powerful graphics facilities without sacrificing DATATRIEVE'S powerful data retrieval facilities. For example, you might use DATATRIEVE to form a subset of data from an annual report and then pass this data on to DECgraph for forming into a multicolored display.

Using VAX DATATRIEVE with DECnet provides distributed access to data.

VAX DBMS (Database Management System)

VAX DBMS provides sophisticated capabilities for creating, accessing, and maintaining large engineering, scientific, and commercial CODASYL-style databases. The major advantages of VAX DBMS are its efficiency and its ability to control the sharing of the same data by a large number of concurrent users.

The definitions that form the logical structure of a VAX DBMS database are stored in the CDD.

You can use VAX DATATRIEVE to manipulate the data stored in a VAX DBMS database. You can use VAX DATATRIEVE to:

- Format into a report or graph data retrieved from a VAX DBMS database
- Link data stored in a VAX DBMS database with data stored in conventional files
- Automatically search the relationships stored in a VAX DBMS database and retrieve data without requiring the user to enter detailed information about the database structure

VAX Rdb/VMS (Relational Database Management System)

VAX Rdb/VMS provides facilities for creating, accessing, and maintaining relational databases. In a relational database, data is stored in the form of two-dimensional tables instead of in complex hierarchies or networks. VAX Rdb/VMS provides all the advantages of a full-feature database management system, including data security and optimized data access. At the same time, it provides the ease-of-use features inherent with the relational-style database:

- A relational database is easy to understand.
- A relational database can be created, modified, and maintained without the services of a professional database administrator.

VAX Rdb/VMS data definitions can be stored in the CDD on an optional basis. As with a VAX DBMS database, you can use VAX DATATRIEVE to retrieve data from a VAX Rdb/VMS database. You can format this data into graphs or reports with VAX DATATRIEVE, or you can send it to the more sophisticated facilities of VAX DECgraph.

VAX TDMS and VAX FMS

VAX TDMS, Digital's Terminal Data Management System, is a programmer productivity tool designed to reduce the high lifecycle costs of developing and maintaining forms-intensive terminal applications on VAX/VMS systems.

TDMS offers a wide range of features making it easy to develop applications that display and collect information, relieving the programmer of many of the "burdens" associated with conventional forms-based applications. TDMS is a part of the VAX Information Architecture, as it uses the VAX Common Data Dictionary and can interact with VAX DATATRIEVE, VAX DBMS, VAX Rdb/VMS, and VAX ACMS. It provides a record-level interface, while VAX FMS offers a field-level interface.

The VAX Forms Management System, FMS, is an alternative to TDMS. It, too, is designed to aid in the development of application programs that use video forms, but takes a different approach from that of TDMS. These two alternative solutions are compared and contrasted at some length in Chapter 7.

VAX ACMS

VAX ACMS, the VAX Application Control and Management System, was designed to reduce the lifecycle costs involved in designing, developing, maintaining and controlling transaction processing and other complex VAX/VMS applications.

Unlike traditional application development tools, ACMS allows for the replacement of large amounts of application code with high level definitions stored in the VAX Common Data Dictionary. With the use of such definitions, users now have available a fourth generation-like language facility that can significantly reduce the development and maintenance lifecycle costs of large, complex software projects.

VAX DECgraph

VAX DECgraph is the VAX Information Architecture's interactive, menu-driven tool for generating graphs from data. It is designed to be used by experienced computer users and novices alike, offering a wide spectrum of capabilities for producing professional quality graphs.

DECgraph is for anyone with a need to see or show data pictorially. With DECgraph, the capability now exists for such people to generate graphs on their own, quickly and easily. Even the most novice of computer users can now become an accomplished designer of graphs in as little as one hour.

DECgraph, then, should be thought of as a productivity tool for anyone who either makes decisions based on data or is responsible for presenting that data to those decision makers. It allows the professional who "owns" a certain set of data, and is therefore "closest" to it in understanding, to create a graph of that data for maximum effect.

VAX DECslide

VAX DECslide is a menu-driven graphic presentation design tool that runs on the VAX/VMS operating system. It is intended for use by anyone who needs to prepare professional-quality presentations and reports.

Anyone with responsibility for creating text and graphic materials for presentations or reports can benefit from using VAX DECslide.

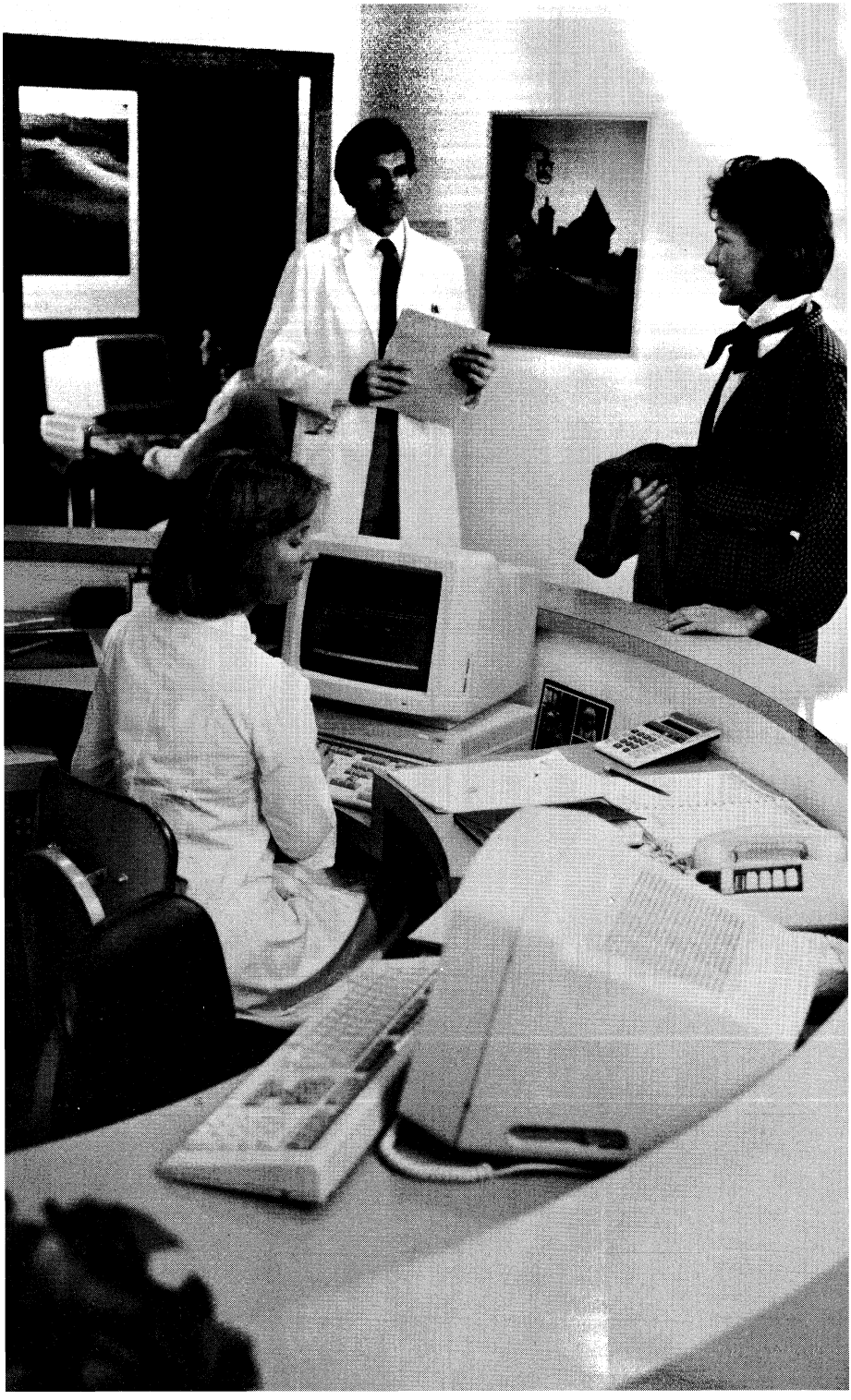
As a layered software product on the VAX/VMS operating system, DECslide runs on any valid VAX/VMS configuration. It can be an effective solution not only in the business world, but also in scientific, educational, manufacturing, and governmental environments as well.

VAX VTX

VAX VTX is a videotex system designed for internal use by businesses and other private organizations. It is a VAX/VMS software product which conforms to international standards for videotex systems; it requires no specialized hardware. VAX VTX delivers information quickly and simply to a variety of computer terminals and personal computers. It can be integrated into Digital's ALL-IN-1 office automation system without any additional hardware. It can also be added as an extremely easy-to-use front end to an organization's transaction processing applications without requiring major rewriting of existing systems.

Organizations that want to present some or all of their internal publications and transaction processing applications simply, uniformly, and efficiently will benefit from VAX VTX. Whether the organization's information is on a single computer or distributed worldwide over a computer network, VAX VTX offers fast, up-to-date information to users.

Using VAX VTX, a department head can make frequently requested and frequently modified departmental publications available from a single source, make them easy to keep current, and keep them readily available for widespread use. Employees who wants quick, simple, reliable access to a variety of information (applications ranging from weather reports, airline reservations, and motel information to personnel policies and procedures, sales information, directories, employee activities, credit union transactions) have a convenient source that saves time and effort.



Chapter 2 ▪ Case Studies in Success with the VAX Information Architecture

In this chapter, you'll find examples of how the VAX Information Architecture products can be used to design a number of business, commercial, and scientific applications. While the individuals and companies are fictitious, they represent the kinds of ways that Digital's information management strategy is being put to work today.

▪ A Relational Solution for Building an OEM's Publishing System

Scribe Industries is an original equipment manufacturer (OEM) that provides a complete system to create, edit, and maintain sophisticated technical documentation sets for the telecommunications industry. Scribe's customers produce a variety of satellite and telecommunications equipment that require highly detailed information to be used by engineering and manufacturing departments in many facilities. It is both essential and expensive for Scribe's customers to keep thousands of pages of documentation up to date.

Because many sections in the documentation sets are related or identical, Scribe's goal was to create a system that could exunderstand and work with the diagrams, specifications, FCC and government regulations, supplier information, and thousands of pages of text, with a minimum of redundancy. For example, before the Scribe system, changing a part description when an identical part is used in five different products involved five separate operations. The new system puts documentation sets into a reliable, easy-to-maintain, and readily available system.

The design of this system required careful planning and analysis of an array of products from many software and hardware manufacturers. The hardware and software used by Scribe had to meet a variety of criteria — factors like price, performance, reliability, software compatibility, and availability of programming languages. The vendor selected had to be reliable and reputable, because once a commitment was made, hundreds of thousands of dollars and years of development would be on the line.

VAX Information Architecture Relational Products Bring Unique Features to Application Development Problems.

James Richards, senior development manager for Scribe, chose Digital's VAX Information Architecture products and VAX systems to design and implement the new system. It was clear to Richards that the most efficient design for the system would be achieved by using a relational database system. Because much of the equipment described in the documentation sets share the same or similar components, and because suppliers of the components must conform to the same regulations, a system that could use relations to make the correlations among and within various documentation sets was essential.

His plan was to develop a distributed workstation approach to handle the editing functions with a local area network of graphics and editing workstations accessing a central database. By using specialized workstations, computer overhead could be greatly reduced, and the user interface could be optimized for a specific task. Several devices had to be supported by the system to accommodate these specialized publication-quality graphics and word processing terminals. Richards also had to plan for the possibility of connecting the system to sophisticated layout and printing systems.

He looked at a number of relational database products and decided Digital's VAX Rdb relational database management systems offered unique features to simplify the design and implementation of the system. That the databases and database applications could run on a complete line of processors of varying sizes was extremely important. As an OEM, having one product that doesn't need to be modified for customers with different system requirements means substantial development savings.

A Choice of Operating Environments to Meet Special Needs

When he looked closer, Richards found that Digital offered something no other vendor did — a choice of operating environments that can share common applications. And VAX Rdb relational systems that can handle the unstructured data types — the graphics, documents, tables, and charts — that make up the lion's share of documentation and manuals.

This choice meant Richards could use the VAX Rdb/ELN database system running under the VAXELN operating environment for the workstations. VAXELN is Digital's realtime operating environment for time critical and dedicated, fixed function applications. The less expensive VAX Rdb/ELN was well-suited to this purpose because the workstations would only be needed to access the central database, update the information using the specialized editors that reside on each one, and then return the updated information to the main system. The database, running VAX Rdb/VMS under the VMS operating system would provide security for the sensitive information in the database.

One reason Richards chose the VAXELN environment for the workstations was that it includes special services that simplify the design of device drivers. So the coding needed to control the unique editing and graphics terminals was easier to write. Richards realized that because the work is primarily clerical, the workstation operators would not need to use the Digital command language, mail facilities, programming services, and other features of VMS.

With Digital Products Price, Performance, and Versatility Make You More Productive.

Richards selected VAX-11/730 computers to serve as workstations connected by a DECnet-VAX local area network to the database residing on a VAX-11/750. The central database is a VAX Rdb/VMS database designed to be under the control of a system manager who can restructure the database dynamically and maintain the security procedures.

Another big benefit of the VAX Rdb approach has been the low cost for Scribe. Richards was surprised at the reasonable price of VAX Rdb/VMS and VAX Rdb/ELN software. Because of the precompilers in the VAX Rdb systems, development of specialized software took less time than was expected. A more economical system developed in less time than originally thought necessary meant good business for Scribe.

With a VAX Rdb/VMS Database, other VMS Layered Products can Use Database Information.

Because the main database resides on a system running VMS, Richards is designing an option for the system that gives users who already have VAXes a way to add the documentation system to a broader VAX network and query the database from an ALL-IN-1 office menu. This allows the administrative parts of the customer's organization to access portions of the database that may be useful, using VAX DATATRIEVE as an interactive query language.

VAX DATATRIEVE is fully integrated into the ALL-IN-1 menu system, so information extracted from the database as VAX DATATRIEVE reports can be incorporated easily into word processing documents. For example, information about suppliers and part ordering may be as important to the accounting department as it is to design and manufacturing. Scribe already has many orders for this option.

▪ A System That Grows . . . from Office Automation to Information Management

For a number of years, the Worldwide Insurance company had been using Digital's ALL-IN-1 office menu system in their corporate offices. Corporate MIS director, Anita Johnson, chose ALL-IN-1 as a way to provide Worldwide with convenient word processing, calendar management, and powerful electronic mail facilities. The system was primarily considered to be a secretarial and management support tool.

Although Worldwide had an extensive corporate data processing department, Johnson found many other departments were using the ALL-IN-1 Form Development tools, VAX FMS, along with VAX DATATRIEVE from the information management subsystem, to respond quickly to many departmental applications needs. For example, when the claims department needed a convenient way to keep track of and report on approved automobile repair shops, Toni Reardon, the department manager, developed the application in just a few weeks. And in only a few days she trained the claims supervisors to maintain the system. The rest of the claims department learned to use simple VAX DATATRIEVE queries in a few hours.

Even in her own department, some developers used VAX programming languages, like VAX COBOL, to write sophisticated programs to help the accounting department meet urgent needs. These programs were incorporated into the ALL-IN-1 profession specific subsystem.

The more Worldwide's office staff used ALL-IN-1, the more they demanded of ALL-IN-1. After the first six months, Johnson sent out a survey to find out how various departments were using the system. She learned that Worldwide's training department designed training materials and overheads with the word processing editor, so she added VAX DECgraph and VAX DECslide business graphics presentation products to provide a convenient, economical way to create slides and overheads for training literature and seminars.

But the major finding was that every department wanted more applications. Many of the programs designed by one department could be used by others. It also became clear that over half of the reports and documents created using the ALL-IN-1 system ultimately ended up in Worldwide's 250 branch offices.

Create a Distributed Network to Manage and Share Information . . . without Forfeiting Your Hardware and Software Investment.

Johnson faced a dilemma. It was clear more resources were needed to fulfill the requirements of Worldwide's users. It was also clear that the move to a more comprehensive information management system would involve a capital investment in hardware and software. What would happen to the VAX-11/750 and all the applications that were now running under ALL-IN-1? Developing new systems at the expense of old ones would be costly and hard to justify.

Clearly one approach was to tie all the departments into Worldwide's mainframe system. But the corporate mainframe was already overloaded handling calculations of actuarial data and maintaining corporate accounting and billing. The communications costs to connect individual terminals to a centralized mainframe were staggering. And the expected response times were inadequate.

Worldwide needed a system with exceptional networking to get the information to its branches quickly. It needed a system that allowed people to be close to the information they provide and maintain. And it needed to preserve its investment in hardware, software, and training.

The answer was to combine the VAX FMS, VAX DATATRIEVE and VAX language applications from the ALL-IN-1 system with VAX VTX business videotex. The key to the system would be to share applications and information throughout the company. With integrated information management products, Worldwide could implement a complete solution. And, because application compatibility through common architectures is the VAX product strategy, Johnson knew that Worldwide would be able to add new products as they were needed.

The VAX Information Architecture Lets You Put the Right Applications in the Right Places.

The real beauty of this VAX Information Architecture approach is the way it takes advantage of the existing programs and systems. For example, with the VAX VTX system Worldwide was able to continue using the same VAX DATATRIEVE program to create and maintain the repair shop report. Johnson wrote a simple VAX DATATRIEVE procedure that puts this report into a format that can be easily incorporated into the VAX VTXinformation base using the ALL-IN-1 editor and the VAX VTX tool for information providers.

The reports on approved repair and appraisal locations, boilerplate for policy forms, premium information, rate structures — even individual policy-holder profiles — could be stored easily in VAX VTX business videotex systems. With powerful distributed capabilities, VAX VTX allows users to access videotex information located on any computer in a network as if the information were on the same system. And because VAX VTX is extremely easy to use, office staff members and agents throughout the country can learn to use it in just a few minutes.

Worldwide's VAX VTX system brings up-to-date information to over 300 people in offices throughout the United States. A network of VAXes and MicroVAXes distributed in 12 regional offices runs VAX VTX. If the international division decides to join the VAX VTX network, the system can handle thousands of additional users by adding more VAXes. And Worldwide insurance agents can get videotex information using Digital personal computers or any one of hundreds of other personal computer models that have the ability to act as Digital VT-100 terminals.

For example, the Chicago office has a VAX-11/730 computer running the portions of VAX VTX used to access, create and maintain information. People in the Chicago office use the system to get current repair shop information from the VAX-11/750 computer in the Indianapolis home office by selecting simple menu items. VAX VTX locates and displays the report in seconds. And the the location of the system with the information makes no difference to the agent in Chicago. When a new shop is added, or if some information about the shop changes, a secretary in the Chicago office modifies the report using the ALL-IN-1 editor and the VAX VTX information provider assistance tool.

When an insurance agent in Springfield, Illinois needs the same information, the agent can telephone the Chicago computer and see the information on a personal computer. In fact, if a vacationing policy-holder from California needs to check policy information, the agent can phone the same system in Chicago and verify the customer's coverage using the same menus. Any authorized terminal connected to any of the VAXes in a network running VAX VTX can get instantaneous information.

Protecting Your Software Investment with Transportable Applications

The big plus for Johnson and Worldwide has been the way the many branches share applications, whether they're developed by the MIS department or by one of the branches. For example, the Cleveland office uses the information in the policy-holder profile to produce a monthly report of upcoming birth dates that affect rates and premiums. Using this report, the branch sends notices of rate reductions to drivers who reach age 25 and homeowners who are 65 and older. Ideas like this make Worldwide more efficient and improve customer service.

Johnson has also instituted an online newsletter in the VAX VTX information system. Departments and branches keep each other up to date about company and industry developments. A listing of the code for applications being used across the country, like the one written by the Cleveland office, are put in the VAX VTX newsletter so other branches can copy the application into their ALL-IN-1 systems.

As Worldwide Grows, so can their Information Management System.
The best part of the VAX Information Architecture strategy is the part that lets a company plan for future growth. That's because in the same way that Worldwide went from an ALL-IN-1 system to a broader information distribution and retrieval system, they can continue to add applications without upsetting current operations. For example, if in the years to come Worldwide needs to add database management, the architecture offers CODASYL and relational systems, VAX DBMS and VAX Rdb/VMS.

The solution for Worldwide was the VAX Information Architecture strategy: a strategy that progresses over time the way people and businesses do so they can respond to new challenges and demands without being penalized for success and the growth that goes with it.

▪ Inventory Tracking in a Manufacturing Facility — Designing a Transaction Processing Application where Time is Critical

Electronic Automotive Suppliers (EAS) is a subcontractor for a major automobile manufacturer. Since 1968 they have been producing electronic ignition systems and pollution control devices for new cars produced in the United States. Today they are facing fierce competition from overseas manufacturers. Plants in Asia and Latin America are attempting to move into the market and are planning to bid aggressively against EAS for a number of major orders.

EAS has sufficient commitments for orders for the next three years. However, it is clear to Howard Conners, the company president, that they will have to substantially lower their bids to win contracts that will keep business running and growing over the next decade. This critical situation caused EAS to carry out a comprehensive study to discover ways to increase productivity and reduce costs.

One key way was to find a reliable, cost-effective solution to inventory control and tracking requirements. Conners estimated that millions of dollars per year could be saved in warehousing, interest expenses, accounting, clerical costs, and manufacturing overtime with a system that would maintain accurate, easily-accessible inventory records and through a material requirements planning system.

Database Management means Thousands of Parts can be Tracked Minute-by-minute.

An inventory tracking system has dozens of people and programs simultaneously entering, updating, and retrieving information about new shipment arrivals, new orders, manufacturing requirements, and accounting procedures. A business can lose time and money if their system cannot share data and so these operations can take place simultaneously. Also, the data used to keep track of the thousands of parts needed to manufacture sophisticated equipment data must be consistent and secure.

Conners called in a consulting firm to evaluate EAS's operation, and recommend solutions. Pete Hall, senior systems analyst with Information Consultants, studied EAS. He concluded that to track parts, produce the simulations needed to do sophisticated planning, and maintain detailed inventories, a database management system was essential.

In fact, without the data independence provided by such a system, it would be impossible to do what EAS required. With a database management system programs can share data definitions so application design is more efficient. The information in the system would be vital if EAS were to make proper parts orders, cash projections, assembly line assignments, and delivery plans. And, only a database management system could offer the safeguards needed to prevent accidental or malicious damage to the data.

An Overall Plan for Efficient Program Design, Easy-to-use Applications, Exceptional Security, and Reduced Maintenance Costs

Hall called for a transaction processing environment to monitor the manufacturing, delivery, and shipment procedures. This runtime environment would include specialized equipment that monitors the number and type of completed products as they come off the assembly line and immediately updates the database with inventory information. Clerical employees, completely unfamiliar with computers, would enter the orders, shipments, and arrivals of parts.

For this reason, Hall planned for the workers to use menus and forms to enter and retrieve information. Analysts would also use menus and forms to enter variables and call programs that model materials demands for EAS products. In all, Hall expected that between 20 and 30 people would use the system, but Conners felt confident that if they could compete successfully, EAS could double in size within five years.

When Hall began his development planning, he came to a roadblock. With the number of senior developers he had available it appeared impossible to complete a job this size in less than five years. By then it would be too late for EAS. He was also convinced that no more than a year could be saved even if he were to take a more expensive approach by setting aside the less important parts of the plan for later.

VAX ACMS Simplifies and Speeds the Development of Large-scale Transaction Processing applications.

Hall discussed EAS's situation with other analysts at Information Consultants. One of his colleagues knew about a recently-developed set of products specifically created for transaction processing applications. It included start-up and recovery instructions for the database management system, integrated menus, and special application development tools to reduce overall development time. These same application development tools also allowed less experienced programmers write more of the code.

If this product was compatible with the proper hardware and came from a reliable vendor, Hall thought it might be a solution for EAS. When he found out the product was VAX ACMS from Digital, he felt even more secure since he was already familiar with VAX Information Architecture information management products. He had planned to use VAX DBMS to design the database management system and his experience with VAX computers convinced him of their versatility and flexibility. The VMS operating system was the premier environment for applications development, having the a wide choice of programming languages and the most complete range of services for programmers and database designers.

Hall scheduled a series of meetings with specialists from Digital's Software Services department. The group believed the project could be done within the three years. Because VAX ACMS provides a development structure that enforces a consistent method of programming, it would make it easier to divide the work among more programmers. Some could work on processing routines to be written in VAX FORTRAN while other programmers wrote VAX ACMS routines. Because VAX ACMS is a "fourth-generation" tool it replaces many hours of programming time with high-level definitions that can be used by less-experienced programmers. With VAX ACMS comprising the body of the transaction and VAX TDMS managing the terminal I/O including the forms and menus, a major portion of the VAX FORTRAN development is eliminated.

The runtime environment is controlled by VAX ACMS. Security procedures, I/O requests, system commands, and system resource allocation are all handled by VAX ACMS. This means much more efficient use would be made of the processor selected. Even with the accelerated schedule, costs would not be over budget. And because the VAX ACMS runtime environment shares resources among users, as more users are added less resources have to be added. So the system will easily handle Conners plans to double usage in future years — without carrying the expense of an underused system in the interim.

A System that's Completed on Time with the Ability to Grow

Using VAX ACMS, VAX TDMS, VAX DBMS, VAX FORTRAN and a VAX-11/780, Information Consultants completed the system in budget and on time. The modeling routines let managers use forms to enter information. The system uses the latest production and inventory data to project how EAS can make the best use of its resources. Product status is closely monitored so contract obligations are being met in record time. Back-ordered goods and over-stocked items have virtually disappeared from EAS's books.

Conners has been able to lower his bids and has secured contracts that will take EAS into the next decade. He is now looking into bids for small parts used in the appliance industry, confident that the VAX ACMS-developed system will be able to grow as the business demands.

▪ Database Support for Cost-Effective Pharmaceutical Research

In the consumer pharmaceutical marketplace, the development of a safe, new over-the-counter pain reliever can add millions of dollars to a company's revenues. Household Laboratories is a large producer of health care products with research facilities located in various cities across the United States. Household has discovered a safe, effective, moderately priced analgesic compound which it hopes to market as an alternative to aspirin and acetaminophen.

The problem Household faces is that the compound is only available in liquid form and the syrup has an extremely unappetizing taste and odor. The marketing department has made it clear that there is no possibility for any commercial success until a tablet or capsule formula can be found. A research team of six scientists headed by Dr. John Colburn has been assigned to find a solution.

The System Speeds the Solution.
Colburn's team must deal with a number of issues. Detailed records must be kept to meet FDA requirements and because all the results will be correlated in ways that may not be obvious when the experiments are actually taking place. It is not unusual, while research is ongoing, for information on other drugs to be released that can influence future test techniques. Because the analgesic being tested by Household has properties similar to other drugs, studies revealing, for example, allergic reactions to other drugs can affect the kinds of analyses to which data will be subjected. Therefore, every aspect of the experiments carried out must be observed and logged.

Household provides research teams with small computer systems to keep track of experiment results. Colburn's team will use a VAX-11/725, a small multiuser system that is ideally suited to workplaces that do not have special electrical or environmental equipment. They will store the results using a VAX Rdb/VMS database, the same relational database management system used by dozens of other research teams working on projects for Household. Colburn can use the system in his research lab to access and update any of these databases with the results of his findings. And other researchers at Household can make use of Colburn's results.

There is also a corporate research library with tables of chemical property characteristics, an encyclopedia of past chemical research, and other pertinent information on a VAX-11/785 at the corporate headquarters. This, too, is accessible to any of the other VAX Rdb/VMS systems.

Using VAX Information Architecture Products to Make the Job Easier.
Colburn's group will use the VAX Rdb/VMS database along with two other VAX Information Architecture products to make the collection and analysis of the data simpler and more efficient. They have designed a series of terminal forms that laboratory technicians will use to record results. Using VAX TDMS-developed screens, technicians will enter data concerning the compound's reactions to being combined with a variety of inert stabilizers and its effectiveness when combined with other compounds. Results will be entered on a regular basis over a number of months.

With VAX DATATRIEVE, Colburn's team will be able to query the database interactively, make correlations among various experiments, and produce reports, charts, and graphs that may be required by the FDA. Most of the researchers have been familiar with VAX DATATRIEVE for years. They've used its English-like commands for data management and applications development before with VAX RMS and VAX DBMS files. They will be able to check results of experiments in progress, so promising experiments can be continued, modified, and complemented, while unsuccessful ones can be discontinued.

Better Results for Faster, Safer, more Profitable Solutions

Colburn's team has used VAX Rdb/VMS to narrow the field of possibilities down to six compounds in under nine months. Using the data stored in the relational database, they will now begin the comprehensive tests needed to assure Household and the FDA of the safety and effectiveness of the new analgesic compound in tablet form. Results of the next two or three years of tests will continue to be stored and analyzed using the VAX Rdb/VMS database.

If the test results are successful, Household expects to have an extremely valuable, safety-tested product available for the general public within a reasonable time. By making the information management resources of VAX Information Architecture products available to laboratory research teams, Household has been able to save money and time while increasing the quality and accuracy of their research procedures.

The stories in Chapter Two of this handbook are fictitious. Any similarity between these stories and actual persons, places, or companies, existing now or in the past, is purely coincidental.


G.14

Chapter 3 ▪ The VAX Common Data Dictionary (CDD)

The *VAX Common Data Dictionary* (CDD) is the central repository for data descriptions and definitions used by other VAX Information Architecture products, and by programs and applications written in any of several of the VAX high-level languages. It is essentially a means of storing, maintaining, and controlling data about data.

Using the CDD, you can

- Create shareable definitions with a common data definition language (CDDL) that can be understood by many VAX programming language compilers and VAX Information Architecture products.
- Modify data definitions in the dictionary without having to edit the programs and procedures using the definitions.
- Specify which users have access to individual definitions, using thirteen separate access privileges and four possible user identification criteria.
- Copy definitions at compile time into a program written in one of many VAX programming languages, including VAX COBOL, VAX BASIC, VAX PL/1 and VAX DIBOL.
- Document the use of a particular definition by making entries into the definition's history list.
- Maintain an area of the dictionary for the storage of data definitions for individual use.

▪ Who Uses the VAX Common Data Dictionary

The VAX CDD is required for all users of VAX DATATRIEVE, VAX DBMS, VAX TDMS, and VAX ACMS. It is optional for users of VAX Rdb/VMS (see Chapter 6).

The CDD is also intended for users who need consistency of data definition, the sharing of definitions across several languages, and centralized storage of common data definitions outside the application programs that use them.

▪ Benefits of the VAX Common Data Dictionary

- Storing data definitions within the CDD eliminates the need to define data within application programs. This applies to application programs written in VAX BASIC, VAX DIBOL, VAX COBOL, and VAX PL/I.
- Storing data definitions in a central area reduces redundancy (multiple copies of the same data definitions) and inconsistency. To change a data definition that affects several application programs, you need to make the change only once, in the CDD, and then recompile the affected programs.
- Multiple programs, although written in different languages, can nonetheless share one or more definitions stored in the CDD.
- The VAX CDD hierarchical model allows different users to organize separately owned portions of the CDD according to their own needs.
- The CDD Data Definition Language utility (CDDL) allows you to define and store shareable record descriptions.
- You have the capability to store portions of the CDD on different devices, or offline.
- Using the VAX CDD history list feature, you can keep a record of each access to a directory or a dictionary object.
- The Dictionary Management Utility (DMU) allows you to create, back up, copy, and protect the CDD hierarchy.
- The VAX/VMS Lock Manager facility allows users to access the VAX CDD concurrently without interfering with one another.
- The Verify/Fix utility allows you to check the integrity of a dictionary file and to fix files that might have become corrupted.

▪ A Closer Look at the VAX Common Data Dictionary

Using the VAX CDD involves the following

- Creating a dictionary hierarchy
- Creating and storing data definitions
- Controlling access to definitions
- Assessing the impact of changing data definitions
- Modifying existing data definitions
- Locating the correct definition to use in an application program
- Copying definitions into application programs
- Maintaining dictionary files

Creating a Dictionary Hierarchy

The VAX Common Data Dictionary is organized as a hierarchy (reversed tree structure) of *dictionary directories* and *dictionary objects*. Dictionary directories are used to organize information within the hierarchy. Dictionary objects are located at the ends of the branches on the hierarchy, and contain the data definitions stored in the dictionary.

These definitions include record descriptions that can be copied into application programs, as well as VAX DATATRIEVE domains, record definitions, procedures, tables, and plots. VAX DBMS schemas, subschemas, storage schemas, and security schemas are also stored in the CDD, as are VAX TDMS and VAX ACMS definitions and, optionally, VAX Rdb/VMS definitions.

When you first install the CDD, one dictionary file named CDD.DIC is automatically created in which the directory hierarchy is physically stored. However, the CDD allows you to store portions of this single logical hierarchy in separate physical files called subdictionary files. The directories that point to separate subdictionary files are called *subdictionaries*.

You create dictionary and subdictionary directories with the *Dictionary Management Utility (DMU)*.

Figure 3-1 illustrates a sample dictionary hierarchy created using DMU commands.

At the top of the sample directory in Figure 3-1 is CDD$TOP, which is called the *root dictionary directory*. The next level consists of four directories — PRODUCTION, CORPORATE, PERSONNEL, and SALES. Three of these directories have *children*.

The directory CORPORATE is the parent of PRODUCT_INVENTORY, ADDRESS_RECORD, and EMPLOYEE_LIST, which are dictionary objects containing record definitions.

The directory SALES also is the parent of three children; but one of these children, JONES, is not a dictionary object. It is a directory, which itself is the parent of a single dictionary object, LEADS_RECORD. SALES is an *ancestor* of the dictionary object LEADS_RECORD.

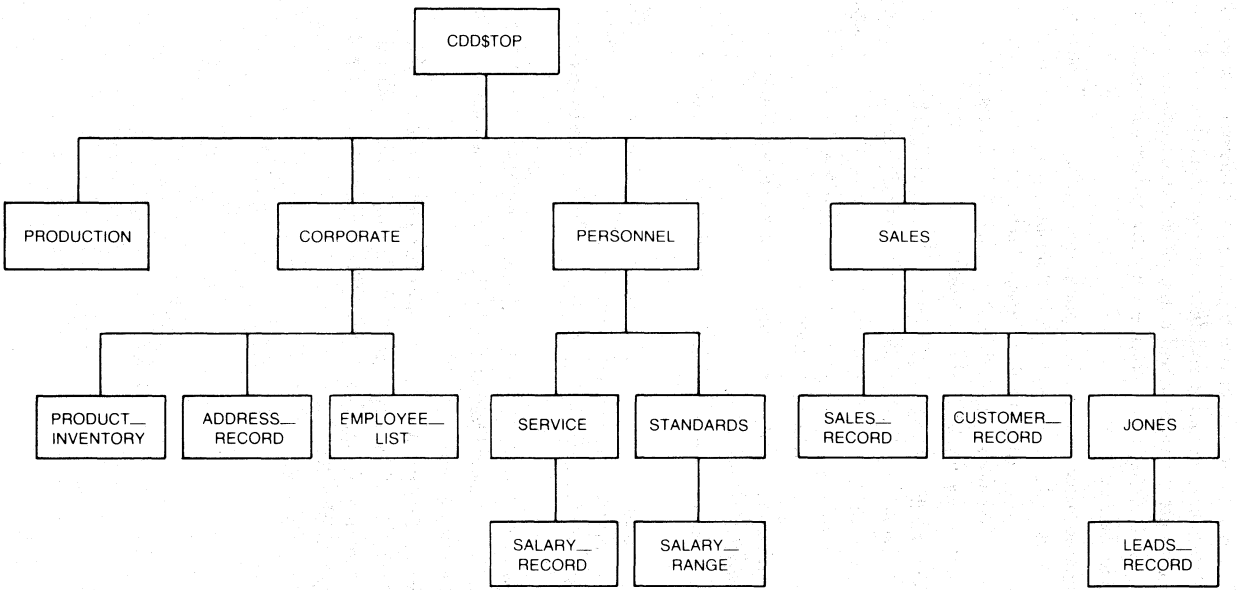


Figure 3-1 ▪ Sample Dictionary Hierarchy

Creating and Storing Data Definitions

You may use the Common Data Dictionary Data Definition Language Utility (CDDL) to create data definitions and store them in the CDD. CDDL provides a generic language that enables you to define records that are understandable to several of the VAXprogramming languages, as well as to the other information management products in the VAX Information Architecture family.

Data definitions are stored as dictionary objects in the dictionary hierarchy.

To create a record definition and insert it into the dictionary, you first create a CDDL source file using VAX EDT or some other text editor. Then you submit this source file to the CDDL compiler, which inserts the record definition into the dictionary.

CDDL source files contain DEFINE and END statements, DESCRIPTION statements, field description statements, and field attribute clauses.

- THE DEFINE STATEMENT

  You use a DEFINE Statement to name a record definition. The name you enter is the *path name* of the definition. (You need not specify the whole path name at this point — the given name or the relative path name is sufficient.) The last name of the path name (in the example, EMPLOYEE_LIST) is the *given name* of the definition; the rest of the path name is the path of directories to the object with the given name (that is, the route the system must navigate through the hierarchy to reach the definition). In other words, in just one step you name an object and specify its place in the dictionary.

- THE DESCRIPTION STATEMENT

  You can use the DESCRIPTION statement to document a record definition or any individual field within a record definition.

- FIELD DESCRIPTION STATEMENTS

  You use the field description statement to describe the field characteristics of a record. A field description statement includes the names of the fields and their data types, as well as other information. CDDL supports four different kinds of field statements:

  - *Elementary field description statements* are used to describe fields that are not subdivided into other fields. The ID field in EMPLOYEE.DDL is an example of an elementary field description statement.
  - *STRUCTURE field description statements* are used to describe fields that are divided into one or more subordinate fields.
  - *COPY field description statements* copy the contents of a record definition into a field of another record definition. This feature ensures that certain commonly used fields are defined in the same way in every record definition that uses them.

- *VARIANTS field description statements* provide alternative descriptions for the same portion of a record.

- FIELD ATTRIBUTE CLAUSES

  You use field attribute clauses to describe characteristics of the fields in a record. There are two kinds of field attribute clauses — general and facility-specific.

  General field attribute clauses describe the storage of data definitions in the CDD. All language processors that use the CDD recognize these attributes.

  Facility-specific attribute clauses describe characteristics of a data definition that affect the interpretation of a record definition by particular language processors. Processors not supporting a particular facility-specific attribute clause ignore it. This feature allows you to tailor a characteristic of a record definition for a particular processor without making it unacceptable to another language processor.

Controlling Access To Data Definitions

The CDD provides several security mechanisms you can use to protect a dictionary against unauthorized access or use.

- THE ACCESS CONTROL LIST

  The CDD restricts access to any particular dictionary directory or object by means of an *access control list* (ACL). When a user attempts to access a particular directory or object, the CDD checks the access control list to determine if this user has the necessary privilege. If the user does not have the necessary privilege, access is denied.

  You use the DMU SET PROTECTION command to set privileges in either of two ways. You can enter a command line such as the following:

```
DMU> SET PROTECTION/POSITION=2/USERNAME=JONES/GRANT=<HS>
     CDD$TOP.SALES
```

  This statement grants privileges HISTORY and SEE (see below) to directory CDD$TOP.SALES for user Jones at the second position in the ACL.

  Or, if you have a VT100- or VT200-family terminal or a VT52, you can enter a SET PROTECTION/EDIT command and then use a screen form and keypad commands.

  There are 13 privileges governing access to a directory or object. Nine of these are CDD privileges, and four are VAX DATATRIEVE access rights. Table 3-1 gives a brief description of each of these privileges.

Table 3-1 ▪ Access Privileges

Privilege	Description
CONTROL (C)	Allows you to read, modify, and delete access control list entries.
DTR_EXTEND/EXECUTE (E)	Allows you to ready a VAX DATATRIEVE domain for EXTEND access, to access a VAX DATATRIEVE table, and to execute a VAX DATATRIEVE procedure.
DTR_MODIFY (M)	Allows you to ready a VAX DATATRIEVE domain for READ and MODIFY access.
DTR_READ (R)	Allows you to ready a VAX DATATRIEVE domain for READ access.
DTR_WRITE (W)	Allows you to ready a VAX DATATRIEVE domain for READ, WRITE, MODIFY, and EXTEND access.
EXTEND (X)	Allows you to create children of dictionary directories and subdictionaries.
FORWARD (F)	Allows you to create subdictionary files.
GLOBAL_DELETE (G)	Allows you to delete dictionary directories and subdictionaries, including any children they may have, with a single command.
HISTORY (H)	Allows you to add entries to history lists.
LOCAL_DELETE (D)	Allows you to delete dictionary objects, as well as directories and subdictionaries with no children; to edit VAX DATATRIEVE procedures; and to replace or recompile data definitions stored in the CDD.
PASS_THRU (P)	Allows you to use a dictionary directory, subdictionary, or object in a path name. You cannot deny yourself PASS_THRU privilege.
SEE (S)	Allows you to see the definition of a dictionary object.
UPDATE (U)	Allows you to update the definition of a dictionary object.

You tailor access to a directory or an object for a particular user by specifying none, some, or all of the privileges above.

- ASSIGNING PRIVILEGES BY INHERITANCE

  Because the CDD is a hierarchical structure, you can reach a directory or object only by following a particular path from the top of the hierarchy (CDD$TOP). Access control lists take advantage of this fact by allowing you to assign privileges by *inheritance*.

  Inheritance works as follows: By default, the CDD assigns all users nearly all privileges at creation. However, if you specify "/NOACL" either with the DMU CREATE function or using the CDDL, no ACL is created. In that case, users inherit the privileges they had at the parent level.

  If you want some but not all of the privileges to be different at the "child" directory or object, you need include in the access control list for this directory or object only those privileges that differ from those granted at the "parent" directory.

  For example, suppose that you want the three record definitions PRODUCT_INVENTORY, ADDRESS_RECORD, and EMPLOYEE_LIST to have the same access control lists. It is not necessary to create three identical access control lists. Because these three definitions share the same parent, CDD$TOP.CORPORATE, you can create a single access control list at the parent level. This access control list will then apply to all three child record definitions, by inheritance, as well as to the parent, if you have specified "/NOACL."

  If at some future time you decide to change the protection on one of these three record definitions, you would create an access control list for the particular definition that contains only the privileges that differ from those in the access control list for CDD$TOP.CORPORATE.

- USING SUBDICTIONARY DIRECTORIES TO PROVIDE SECURITY

  Using the DMU command CREATE/SUBDICTIONARY, you can create separate dictionary files to hold portions of your CDD. The directory that points to a separate dictionary file is called a subdictionary directory or *subdictionary*.

  Subdictionary files can be stored anywhere: therefore, subdictionaries can be very helpful in the following ways.

  - You can store sensitive material offline when this material is not being used.
  - You can use VMS file protection, in addition to CDD access control lists, to control access to different dictionary files.
  - You can use one dictionary to serve several distinct organizations, as in a timesharing system, but each organization can have its own subdictionary on its own disk. This allows you to charge each organization for the amount of dictionary space its data descriptions use.

Assessing the Impact of Change: The History List

The *history list* feature of the VAX CDD allows you to document and to monitor the use of each dictionary directory, subdictionary, and object.

Documenting the use of dictionary objects is a very important function of the dictionary. For example, before modifying a record definition, the data administrator needs to know what other definitions are affected and what programs and procedures need to be changed as a result. Also, programmers using the dictionary need to know at a glance the purpose and the contents of a definition they are considering using in an application.

A *history list entry* contains information about user access, including the action taken, the person responsible, the facility used, and the date and time. You can create an entry in a history list of a directory, subdictionary, or object when you

- Create a directory, subdictionary, or object
- Modify a directory, subdictionary, or object
- Modify an access control list
- Copy a directory, subdictionary, or object to another part of the dictionary
- Access an object from a program written in a VAX programming language or from one of the other VAX Information Architecture products

For CDDL, DMU, and most of the languages using CDD, you use the /AUDIT qualifier to create a history list entry. You can add your own text to the information automatically stored in a history list entry by including your comment in quotation marks.

Modifying Data Definitions

Because the information needs of organizations change, it is often necessary to change data definitions as well. To change a data definition, you create a new CDDL source file containing the path name of the record definition you want to replace, and then compile the new source file with the command CDDL/REPLACE.

For example, a corporation decides to change the number of job classifications from 150 to 200, and the number of incremental salary levels from 5 to 4. This business decision necessitates a change in the record definition CDD$TOP.PERSONNEL.SALARY_RANGE in the CDD. The current definition in that location is as follows.

```
DEFINE RECORD CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE
    DESCRIPTION IS
    (This record stores minimum salaries
    for the five incremental salary levels within
    each of 150 job classifications.)
  SALARY_RANGE     ARRAY 150,5
                   DATATYPE IS UNSIGNED NUMERIC
                   SIZE IS 8 DIGITS 2 FRACTIONS.
END SALARY_RANGE RECORD.
```

To implement this policy change, the data administrator creates the source file RANGE2.DDL, and places in it the following record definition.

```
DEFINE RECORD CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE
    DESCRIPTION IS
      (This record stores minimum salaries
      for the four incremental salary levels within
      each of 200 job classifications. It reflects
      Personnel Policy # 4022.)
    SALARY_RANGE     ARRAY 200,4
                     DATATYPE IS UNSIGNED NUMERIC
                     SIZE IS 8 DIGITS 2 FRACTIONS.
END SALARY_RANGE RECORD.
```

Note that this record definition is identical to the former definition except for the size of the array and the added comment in the DESCRIPTION clause. The Data Administrator then enters the following command:

```
$ CDDL/REPLACE/AUDIT="CHANGED TO
     CONFORM TO POLICY #4022" RANGE2.DDL
```

CDDL responds by removing the original CDD$TOP.PERSONNEL.SALARY_RANGE and replacing it with the new definition, keeping the original access control list and history list, and creating a new history list entry documenting the change.

If you modify a template record definition, you should modify all the record definitions that use that template. You do this by determining with the history list all the record definitions that need to be changed, then specifying these definitions to a CDDL/RECOMPILE command.

For example:

```
$ CDDL/RECOMPILE/AUDIT CDD$TOP.CORPORATE.EMPLOYEE_LIST,
$_CDD$TOP.SALES.CUSTOMERRECORD,
     CDD$TOP.SALES.JONES.LEADS_RECORD
```

Locating the Correct Data Definition

Programmers who want to copy record definitions need some way to find the definition they want. Using meaningful names for definitions is helpful, but even so, several definitions often have similar names.

The CDD provides two alternative methods for programmers to easily check the purpose and contents of a record definition. You can either display embedded explanatory text, which has been previously entered to describe particular record definitions, or you can just as easily display the entire source file, which of course contains this explanatory text.

Copying Definitions into Application Programs

A programmer who finds the definition that provides the desired data description can easily include that definition in a program at compile time. For example, a programmer could include the record definition

```
CDD$TOP.CORPORATE.ADDRESS_RECORD
```

in the VAX COBOL program EMPADR.COB by inserting the following command in the WORKING-STORAGE section of the program source file:

```
COPY "CDD$TOP.CORPORATE.ADDRESS_RECORD" FROM DICTIONARY
```

To compile the program, the programmer uses the following DCL command:

```
$ COBOL/AUDIT EMPADR
```

The COBOL compiler retrieves the definition from the CDD and compiles it as COBOL object code. Because the /AUDIT qualifier is used, the COBOL compiler also makes an entry in the history list of ADDRESS_RECORD documenting the operation.

The other VAX programming languages that use CDD record definitions work in a similar manner.

Maintaining Dictionary Files

A dictionary file can experience errors or inconsistencies caused by hardware failures or other uncontrollable events. Because of this, the VAX CDD provides a utility, CDDV, that you can use to check the condition of your dictionary and subdictionary files, and repair them if errors are detected.

You can also use CDDV, when disk space is low, to compress dictionary and subdictionary files, and to return any free space to the operating system for use by other files.

In the following example, the system manager checks the condition of a subdictionary file using the VERIFY command:

```
CDDV> VERIFY MISC$DISK:[MACWIRE.CDD]PRIVATE.DIC
```

CDDV returns a message indicating that the dictionary contains errors.

The system manager therefore attempts to repair the subdictionary by issuing the following command:

```
CDDV> FIX MISC$DISK:[MACWIRE.CDD]PRIVATE.DIC
```

Wherever possible, CDDV recreates directories. When it cannot recreate a directory, it places the descendants of unrecreated dictionaries in a directory called CDD$LOST_NODES.

▪ Using the VAX Common Data Dictionary with Other Software Products

The VAX Common Data Dictionary is the "hub" of the VAX Information Architecture, serving as the central repository of the data definitions used by many of the other VAX Information Architecture products.

VAX DATATRIEVE

All VAX DATATRIEVE domain definitions, record definitions, tables, procedures, plot definitions, and view domains are stored in the CDD.

VAX DBMS

All VAX DBMS schema, subschema, storage schema, record definitions, and security schema definitions are stored in the CDD.

VAX Rdb/VMS

All VAX Rdb/VMS data definitions must be stored in the CDD in order to be used by VAX DATATRIEVE, VAX TDMS and/or VAX ACMS. Otherwise, storage of VAX Rdb/VMS data definitions in the CDD is optional.

VAX TDMS

All VAX TDMS form and request definitions are stored in the CDD.

VAX ACMS

All VAX ACMS task, menu, application, and task group definitions are stored in the CDD.

Languages

In addition, several of the VAX programming languages can access CDD record definitions at compile time. The VAX languages that can use the CDD are

- VAX COBOL
- VAX BASIC
- VAX PL/I
- VAX DIBOL

Using the CDDL, it is possible to specify special conditions required by a particular language compiler without affecting other use of the same definition by other compilers.

Summary

The VAX Common Data Dictionary, then, helps the system manager coordinate an effective data management system; it reduces programming and maintenance costs; and it ties together VAX programming languages and VAX Information Architecture products.

Chapter 4 ▪ VAX DATATRIEVE

VAX DATATRIEVE is an easy-to-use tool for managing and manipulating data either interactively at a terminal or from an applications program. If you know a handful of simple, English-like commands and statements, you can interactively retrieve, store, modify, and sort data, and report on it in meaningful ways.

With VAX DATATRIEVE, you can:

- Create data definitions that can be used to store and retrieve data uniformly, either interactively or from application programs
- Store or modify data in RMS files, VAX DBMS databases, or VAX Rdb databases
- Retrieve data from RMS files, VAX DBMS databases, or VAX Rdb/VMS databases and display the data on a terminal, write it to a data file, or print it on paper
- Produce formatted reports using specified selections of data
- Create pie charts, bar graphs, line graphs, and scatter plots based on specified selections of data
- Use forms to format the terminal screen for data display or data collection
- Use a text editor to correct syntax errors and typing mistakes
- Access RMS, DBMS and Rdb data files located on remote systems
- Call VAX DATATRIEVE functions (including data access) from a program written in a high-level VAX programming language such as COBOL, FORTRAN, or PL/1

▪ Who Uses VAX DATATRIEVE

VAX DATATRIEVE is intended for a broad spectrum of users, including:

- Managers who need ready access to different views of data for decision-making and/or monitoring purposes
- Organizations that need a data storage system that can be run by clerical personnel

- Organizations that need to access data on a distributed network without being concerned with where the data is actually located
- Applications programmers who want to save coding/debugging time and source space by having VAX DATATRIEVE handle such functions as: finding and opening data files, performing input and output operations, formatting data, converting data types, handling error and end-of-file conditions

With its set of simple, English-like statements, VAX DATATRIEVE is designed to be used by people with only modest computer experience as well as by computer professionals. It includes a special tutorial facility called Guide Mode to help any user get started quickly.

▪ Benefits of VAX DATATRIEVE

VAX DATATRIEVE provides the following benefits:

- Extensive use of the VAX CDD, making it unnecessary for you, in DATATRIEVE, to specify locations and definitions as you would with a traditional programming language
- An English-like query language that allows easy access to data stored in RMS files, VAX DBMS databases, and VAX Rdb databases
- An Applications Development Tool (ADT) that provides a simple, interactive means of defining record formats, RMS files, and VAX DATATRIEVE procedures
- A Report Writer that allows you to create formatted reports on any desired selection of data
- A text editor (callable EDT) for easily changing record definitions or correcting syntax errors
- Support for the forms management facilities of VAX FMS and VAX TDMS, which allow you to format the screen for data display or collection purposes
- Graphics support that allows you to create a variety of color or black-and-white charts and plots based on specified selections of data
- A tutorial called Guide Mode that introduces you to VAX DATATRIEVE
- A distributed access facility that allows you to access data stored on remote systems
- Security and protection facilities that allow you to limit access to data definitions and to protect those definitions from corruption

A Closer Look at VAX DATATRIEVE

Using VAX DATATRIEVE involves:

- Using Guide Mode to learn VAX DATATRIEVE
- Creating special data definitions called *domains*, as well as record definitions, and storing them in the *Common Data Dictionary (CDD)*
- Retrieving and displaying data
- Storing and modifying data
- Using the VAX DATATRIEVE editor to correct errors in command lines
- Using relational facilities such as CROSS, which dynamically joins separate files
- Writing and using procedures
- Using the Report Writer facility to create formatted reports
- Using the graphics facility to create charts and graphs
- Calling VAX DATATRIEVE facilities from application programs
- Using forms to format the terminal for data entry and data display

Using Guide Mode to Learn VAX DATATRIEVE

VAX DATATRIEVE provides a tutorial facility called Guide Mode. It is particularly useful to the inexperienced user who wants assistance during a VAX DATATRIEVE session.

You use the SET GUIDE command to invoke Guide Mode. Guide Mode then helps you through a VAX DATATRIEVE session with a series of prompts. At any time in this mode of operation, you can request a list of commands, statements, names, or value expressions that you can enter.

Creating and Storing Domains

When you use VAX DATATRIEVE to manage or manipulate data, the data you work with is made available to you through special data structures called *domains*. A domain relates a data file to a CDD record definition that describes the data fields in that file.

The domain allows you to access data without having to describe the data or specify where it is located. In other words, when you access data with VAX DATATRIEVE, you can often specify the data you want with a single name. In contrast, if you were to use a VAX COBOL program to access the same data, you would need to describe the data to be accessed and specify where it was physically located (device and file name).

Domains are stored as *domain definitions*. In its simplest form, a domain definition consists of the name of the domain, the name of a record definition, and the name of a data file. In effect, then, a domain associates a record definition with a data file that consists of records stored in the same format.

The following is an example of a domain definition:

```
DEFINE DOMAIN PERSONNEL USING
     PERSONNEL_RECORD ON PERSONNEL.DAT;
```

In this example, PERSONNEL_RECORD is a record definition that describes a particular record format, and PERSONNEL.DAT is a VAX/VMS file that contains data stored in the format described in PERSONNEL_RECORD. PERSONNEL is the domain that logically connects the format in PERSONNEL_RECORD with the actual data in PERSONNEL.DAT.

VAX DATATRIEVE allows you to define a domain that points to other domains. Such a domain is called a *view domain*, because it provides a logical view of data stored in more than one data file.

A *remote domain* can also be created. In a remote domain, you can use a record definition and a data file that are both stored on a remote system, linked to the first by DECnet.

VAX DATATRIEVE also allows you to define *DBMS domains*. Using DBMS domains, you can useVAX DATATRIEVE to store, modify, anddisplay data managed by VAX DBMS.

You can create domains and record definitions with the *Application Design Tool (ADT)*. ADT is an interactive tool that prompts you for the information needed to create the definitions for a domain or a record. Using the example of a domain definition, you are prompted for:

- The domain name
- The file specification of the data file
- The name of each field in the file record
- The type of data in each field
- The format for fields containing dates, numbers, or money
- The length of fields containing character strings
- The organization of the data file
- The name and attributes of each index key for indexed files
- The name of a command file to contain the definitions

Should you be unable to answer a given prompt, ADT can suggest answers to you, in non-programming language. You can instruct ADT to create the data file you defined, and to add to your default directory in the CDD the domain and record definitions you created. If you do not do this, ADT creates a command file containing your definitions. You can then create the data file and store your definitions in the CDD any time you wish, by invoking the command file in response to the *DTR>* prompt of VAX DATATRIEVE.

Retrieving and Displaying Data

VAX DATATRIEVE allows you to retrieve and display data in ways that would otherwise require input and output functions performed by a program written in a high-level programming language.

For example, a typical programming language might retrieve and display the records of all employees named Foster using a loop similar to the following:

```
LOOP:
        READ EMPLOYEE-FILE
        AT END EXIT
        IF LAST_NAME NOT = "FOSTER"
        GO TO LOOP
        PRINT FIRST_NAME, LAST_NAME, ADDRESS...
        GO TO LOOP
```

Before writing these lines of code, the programmer would have had to define the record data structures comprising the fields FIRST_NAME, LAST_NAME, ADDRESS and so forth. And the programmer would have had to specify where the record could be found.

In VAX DATATRIEVE, all the work accomplished by the preceding six lines of code can be accomplished by the single line:

```
PRINT EMPLOYEES WITH LAST_NAME = "FOSTER"
```

Of course, the domain EMPLOYEES and the last name FOSTER must have previously been defined.

Using VAX DATATRIEVE, you retrieve and display data interactively as follows:

- Use the READY command to access a particular domain
- Use the FIND command with a *record selection expression (RSE)* to form *collections* of records, and/or
- Use the PRINT command with an RSE to display (and sort) *record streams*

A typical session to retrieve and display data might proceed as follows:

```
DTR> READY PERSONNEL(RET)
```

This command accesses the domain named PERSONNEL.

```
DTR> PRINT FIRST 2 PERSONNEL(RET)
```

In response to this statement, VAX DATATRIEVE uses the record definition contained in the domain PERSONNEL to access and format the data contained in the data file pointed to in domain PERSONNEL; it then displays the following lines on your terminal:

```
                    FIRST       LAST            START              SUP
 ID      STATUS     NAME        NAME     DEPT   DATE      SALARY   ID
00012 EXPERIENCED CHARLOTTE   SPIVA       TOP  12-Sep-1972 $75,892 00012
00891 EXPERIENCED EDWARD      HOWELL      F11   9-Apr-1976 $59,594 00012
```

The headings in this display are the names of the individual fields described in the record definition used to access the data displayed. Note that two records are displayed, as specified in the PRINT statement. If you want to put this information in a file, you can specify an output file in the command line, as follows:

```
DTR> PRINT FIRST 2 PERSONNEL ON FILE.DAT(RET)
```

You can also send the information to a line printer, as follows:

```
DTR> PRINT FIRST 2 PERSONNEL ON LP:(RET)
```

The following statement establishes a collection of records:

```
DTR> FIND PERSONNEL WITH STARTDATE AFTER "01-Jan-1982"(RET)
```

And might yield a response such as:

```
[50 records found]
```

This collection is called the *current* collection. By specifying record selection expressions in additional FIND statements, you can narrow this collection down in any way you wish. For example, the following statement locates four records in the current collection:

```
DTR> FIND CURRENT WITH DEPARTMENT EQUAL
     "SALES" OR "MARKETING" AND ZIP_CODE EQUAL 02138

[4 records found]
```

This collection is now the current collection. If you wish at this point, you can use the PRINT statement to display these records on the terminal screen, or write them to a file, or print them on paper. For example, you could display the data on your terminal screen with the statement:

```
DTR> PRINT ALL NAME, ADDRESS, PHONE(RET)
```

Storing and Modifying Data

You store data interactively as follows:

- Use the READY command with WRITE access to reach a particular domain
- Use the STORE command to store a new record
- Use the STORE command with the REPEAT command to store a specified number of new records

A typical session at the terminal to store new records in a domain might appear as follows:

```
DTR> READY YACHTS WRITE↵
DTR> REPEAT 2 STORE YACHTS↵
Enter MANUFACTURER: SARTINI↵
Enter MODEL: 6800↵
Enter RIG: YAWL↵
Enter LENGTH-OVER-ALL: 40↵
Enter DISPLACEMENT: 20000↵
Enter BEAM: 12↵
Enter PRICE: 82000↵
Enter MANUFACTURER: CARDINAL↵
Enter MODEL: SAVANT↵
Enter RIG: SLOOP↵
Enter LENGTH-OVER-ALL: 27↵
Enter DISPLACEMENT: 7200↵
Enter BEAM: 8↵
Enter PRICE: 23000↵
DTR>
```

You modify data in existing records as follows:

- Use the READY command with MODIFY to access a particular domain
- Use FIND to form a collection of records to be modified
- Use the SELECT command to access a particular record in the collection
- Use the MODIFY command to update or correct any desired field or fields in the selected record

A typical session at the terminal to modify a record might appear as follows:

```
DTR> READY YACHTS MODIFY↵
DTR> FIND YACHTS WITH MANUFACTURER = "CARDINAL"↵
[30 records found]
DTR> SELECT FIRST↵

DTR> PRINT↵
```

```
                                    LENGTH
                                     OVER
MANUFACTURER    MODEL      RIG       ALL    WEIGHT   BEAM   PRICE

 CARDINAL       ADONIS     SLOOP     18        575    08    $1,800

DTR> MODIFY PRICE(RET)
Enter PRICE: 2100(RET)
DTR> PRINT(RET)

                                    LENGTH
                                     OVER
MANUFACTURER    MODEL      RIG       ALL    WEIGHT   BEAM   PRICE

 CARDINAL       ADONIS     SLOOP     18        575    08    $2,100
```

If an entire collection of records is to be modified in the same general way, you can use a MODIFY statement with the keywords ALL OF and an appropriate record selection expression, instead of forming a collection in one step and then using a series of SELECT and MODIFY statements to modify individual records in the collection.

To delete a record, you follow the procedure outlined above, but use an ERASE statement instead of a MODIFY statement.

Using the VAX DATATRIEVE Editor

You can use the VAX DATATRIEVE Editor to edit the last command statement you entered, to save you from reentering the entire command line should you make a mistake or change your mind.

You can also use the Editor to modify domain and record definitions, procedures, and other objects stored in the Common Data Dictionary.

To use the editor, you simply type EDIT followed by a carriage return. VAX DATATRIEVE then places the last command or statement you entered in the Editor's main buffer and on your terminal screen. You can now edit the text or your statement in any way you wish.

To edit an object in the CDD with the VAX DATATRIEVE editor, you type EDIT followed by the name of the dictionary object. You then edit the dictionary object in any way you wish.

Using CROSS to Access Multiple Files

Although not a relational database system, VAX DATATRIEVE has a relational facility for linking a number of domains dynamically. This relational facility is the CROSS clause. Using the DATATRIEVE CROSS clause, you can use a single statement to dynamically *join* data from a number of separate RMS, DBMS and/or Rdb files.

This is an especially powerful facility that makes it possible to join records from different files related to one another by a common field. For example, a college might maintain a Currrent Academic Status file containing student ID numbers and grades and a separate Registration file containing student ID numbers and home addresses. At the time grades are to be mailed home, the with the Registration file to dynamically link grades with home addresses. The VAX DATATRIEVE statement to accomplish this relational operation might appear as follows:

```
PRINT ACADEMICSTATUS CROSS REGISTRATION OVER STUDENT_ID
```

The STUDENT_ID field is the common field that makes linking these two files possible.

Writing and Using Procedures

Using the DEFINE PROCEDURE command, you can define sequences of VAX DATATRIEVE commands and statements and store them in the Common Data Dictionary for later use. These stored sequences of commands and statements are called *procedures*. Procedures can be embedded in other procedures.

They are especially useful in cases where the same set of VAX DATATRIEVE commands is used repeatedly. For example, you could include all the commands necessary to create a weekly inventory report on the same group of products in a procedure that you could then invoke with a single command.

Procedures can be invoked by users at the terminal, or by application programs.

Using the Report Writer Facility to Create Formatted Reports

The VAX DATATRIEVE Report Writer provides a set of formatting options for producing printed reports with page and column headings, page numbers, totals, subtotals, and other summary information.

You can control the format of a report or allow the Report Writer to do some or all of the formatting for you.

The following example illustrates use of the VAX DATATRIEVE Report Writer:

```
DTR> YACHTS WITH LOA = 36RET
[5 records found]
DTR> REPORT YACHTS WITH LOA = 36 SORTED BY BEAM, PRICERET
RW> SET REPORTNAME = "REPORT WITH"/"A SPLIT TITLE"RET
RW> SET COLUMNPAGE = 50RET
RW> PRINT BEAM, BUILDER, DISP, PRICERET
RW> ENDREPORTRET
```

```
                    REPORT WITH          25-JUL-1984
                  A SPLIT TITLE          Page 1

  BEAM      MANUFACTURER        WEIGHT        PRICE


    11        PEARSON           17,700
    11        ISLANDER          13,450       $31,730
    12        CABOT             15,000
    12        ERICSON           16,000
    12        I. TRADER         18,600       $39,500

DTR>
```

Using the Graphics Facility

Using the VAX DATATRIEVE graphics facility, you can create charts and graphs from any desired selection of data. The graphs you can create include bar charts, pie charts, histograms, scatter graphs, time series graphs, and linear regression plots. Figure 4-1 shows a graph created by the VAX DATATRIEVE graphics facility.

To create a graph, you simply use a PLOT statement that includes the name of the graph you want to create and a record selection expression that specifies the data you want to use. For example, you can display a multi-bar chart on a graphics terminal screen with a statement such as:

```
PLOT MULTI_SHADE ALL DATE, SERVICES, EQUIPMENT_SALES,
     REVENUE OF ANNUAL_REPORT
```

The resulting chart shows annual report data in the form of lines with cross-hatching underneath, for trend comparison.

Using a DECwriter IV, an LA50 or an LA100 printer, you can print this chart with the statement:

```
PLOT CROSS-HATCH
```

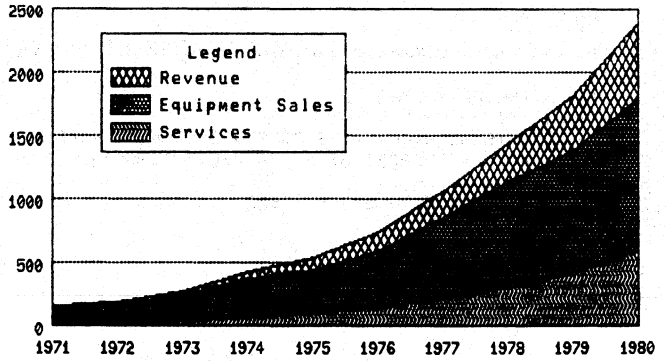


Figure 4-1 ▪ Example of a VAX DATATRIEVE Graph

Calling VAX DATATRIEVE facilities from programs

The VAX DATATRIEVE Call Interface allows you to use the VAX programming languages to call all the VAX DATATRIEVE data management facilities except ADT, Guide Mode, and the VAX DATATRIEVE Editor. Using the Call Interface, programs can pass command lines to VAX DATATRIEVE and receive back records, print lines, and messages.

Using VAX DATATRIEVE to handle standard I/O tasks can greatly increase programmer productivity, because it removes from programmers the burden of having to specify data definitions and data storage locations in their programs. Programs can be shorter, less complicated, and easier to debug and to maintain.

To access data using the Call Interface, you need only pass DATATRIEVE commands and record selection expressions to the Call Interface; you do not need to write code into the program to describe the file structure and record structure of the data to be accessed, or code to optimize the search for the data. At run time, VAX DATATRIEVE automatically locates the correct records.

The Call Interface also allows a program to use VAX DATATRIEVE *tables* and procedures. Tables used by more than one program can be stored in the Common Data Dictionary and accessed through the Call Interface, instead of being included in the programs using them.

You use tables to associate short fields, to act as "code words," with longer fields. For example, a table of zip codes might associate a list of zip codes with the state and town represented by each zip code.

If you use VAX DATATRIEVE to store records, you can take advantage of the automatic validation and default values offered by VAX DATATRIEVE.

VAX DATATRIEVE also handles other functions without the need for language statements. For instance, programmers can use VAX DATATRIEVE to:

- Format data for output
- Convert data types
- Handle errors and conditions such as end-of-file

Programs can also be used to set up a customized interface to VAX DATATRIEVE for the end user and to create new keywords.

Using Forms

The VAX DATATRIEVE Forms Interface provides a convenient method for formatting the terminal for data entry or data display. You can design a form using either VAX FMS or VAX TDMS, insert the form in a form library, and then specify the form and form library names in a VAX DATATRIEVE domain definition.

The following domain definition includes the name and library of a form:

```
DEFINE DOMAIN PERSONNEL_FORM USING PERSONNEL_REC
     ON PERSONNEL.DAT FORM IS PERSON IN FORMSLIB;
```

In this example, VAX DATATRIEVE uses the form PERSON, in form library FORMSLIB, to format the screen for data entry and display. To display the form, you use a statement such as:

```
PRINT PERSONNEL_FORM WITH DEPT = "F11"
```

VAX DATATRIEVE then displays the personnel records of people in Department F11 on a full-screen form.

▪ Using VAX DATATRIEVE with Other Software Products

VAX DATATRIEVE interacts with the following products in the VAX Information Architecture family of data management tools. It also interacts with VAX FMS (Forms Management System).

The VAX Common Data Dictionary (CDD)

VAX DATATRIEVE uses the CDD as a central storage facility for definitions, including domains, records, views, tables, plots and procedures. The CDD is, therefore, a prerequisite for VAX DATATRIEVE.

VAX DBMS and VAX Rdb/VMS

You can use VAX DATATRIEVE as a convenient way to access data managed by either VAX DBMS or VAX Rdb/VMS.

You establish VAX DATATRIEVE access to VAX DBMS or VAX Rdb/VMS data by using DEFINE statements. The DEFINE statements specify the database to be queried, and associate VAX DBMS or VAX Rdb/VMS records with VAX DATATRIEVE domains.

VAX TDMS

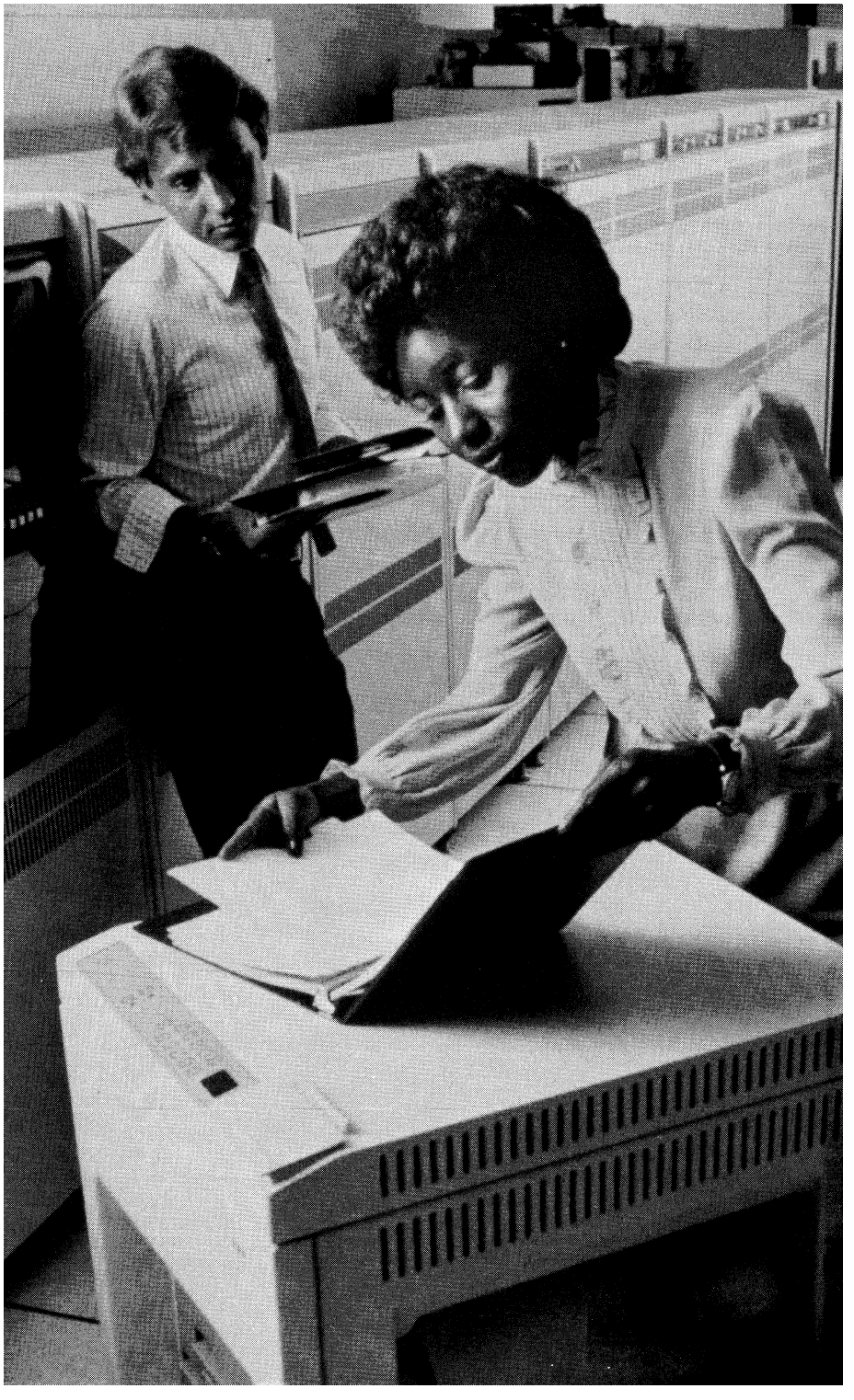
Although you can use VAX DATATRIEVE by itself for terminal I/O, using VAX DATATRIEVE in conjunction with forms created by VAX TDMS has several advantages:

- For many people, forms are less threatening and more manageable than prompts.
- Forms created by VAX TDMS allow you to legibly display complex records on the screen.
- Forms created by VAX TDMS allow you to place help text with the fields or forms being displayed.
- The special highlighting features available with VAX TDMS, such as reverse video and bolding, can make data entry easier and more accurate.

VAX DECgraph

VAX DECgraph allows you to create much more sophisticated graphic displays than are possible with the VAX DATATRIEVE graphics facility, but without sacrificing any of the power and ease of using VAX DATATRIEVE for data access.

Using the VAX DATATRIEVE/DECgraph interface, you can form a data collection with the FIND command, and then pass the data on to DECgraph. You can then use VAX DECgraph to plot the data and allow you to control such characteristics as scaling, labels, and colors.



Chapter 5 ▪ VAX DBMS Database Management System

VAX DBMS is a general purpose, *network-model* database management system that complies with the specifications of the Conference on Data Systems Languages (CODASYL).

With VAX DBMS, you can:

- Create and maintain multiple databases
- Store and retrieve data efficiently
- Separate data definitions from the applications programs that use them
- Centralize all data definitions in the VAX Common Data Dictionary for common use and maintenance
- Define useful relationships between records
- Separate data definition from data storage
- Tailor user views of data
- Centralize the administration of data
- Maintain data integrity and security
- Allow concurrent access to databases

▪ Who Uses VAX DBMS

VAX DBMS is intended to serve as a data management system for organizations in which:

- There is a large amount of data retrieval
- There are many concurrent users
- Data relationships are complex
- Database implementations are long-term
- A professional database designer/administrator is available

▪ Benefits of VAX DBMS

As a CODASYL-style database management system, VAX DBMS allows you to:

- Separate data definitions from the application programs that use those definitions.

  Separating data definition from program development can lower the cost of program development and maintenance.

- Centralize data definitions.

  Centralizing data definitions avoids multiple definitions and makes it easier for many users to share the same data.

- Formally define useful relationships between data items.

  If you were designing an inventory application, for example, you would want the relationship between the parts you keep on hand and the supplier who provides those parts to be represented in the database.

- Separate the physical definition of data from the logical definition of that data.

  Doing this means that the development programmers in your organization need not be concerned with how the data to be manipulated by their programs is actually stored. This in turn saves development costs.

- Tailor different user *views* of data.

  Each program views a database through a special logical structure written and maintained outside the program. Called a *subschema,* it can be thought of as a window into the database.

  Subschemas (that is, tailored views of a database) facilitate *data independence* by allowing you to change data formats without involving programs that are not affected by the changes. You simply create a different view of the database for the affected programs.

- Centralize administration of the database.

  By separating programs from the data definitions used by those programs, you can separate the responsibility for the creation and maintenance of programs from the defining of data and the maintaining of a database. This requires that a *database administrator (DBA)* be available in your organization.

- Maintain database integrity.

  VAX DBMS allows you to use several safeguards to ensure that data is accurate, complete, and timely. These include making backup copies of the database and using *before-image journaling* to roll back a transaction in the event of an abnormal program termination, or *after-image journaling* in the event of a hardware or software failure that corrupts or destroys the database.

- Permit concurrent access.

  Allowing more than one user to access the same database concurrently involves ensuring against conflicts, so that two or more users do not simultaneously attempt to execute similar operations on the same record or records. To protect the database during concurrent use, VAX DBMS uses a system of locks that controls access to the database at various levels.

A Closer Look at VAX DBMS

Using VAX DBMS involves the following:

- Designing databases in accordance with your specific informational needs
- Writing the logical structure of the database (schema) and storing it in the VAX Common Data Dictionary (CDD)
- Creating the physical database
- Storing, retrieving, updating, and deleting data with Database Query (DBQ)
- Ensuring consistency and accuracy
- Using the Database Operator (DBO) to fine-tune the database for best performance
- Accessing database files from application programs
- Working with the other Database Operator (DBO) utilities
- Protecting database security and integrity

Designing Databases

The tasks of analyzing data and designing a VAX DBMS database are usually performed by a *Database Administrator (DBA)*.

The following steps are usually involved:

- Isolating the existing data entities (for example, employees, products manufactured, vendors)
- Plotting the relationships between these data entities, then determining how queries to the database are to be handled

A data entity is further broken down into its component parts, or *attributes*. For example, the data entity EMPLOYEE might have the attributes: name, badge number, social security number, and location.

Data Definition and Storage

Once the data entities that are to compose a database are isolated and their relationships determined, they must be carefully defined. Data entities are defined as *records*. A record consists of individual data *items* that correspond to the attributes of the data entity. For example, a record named EMPLOYEE might consist of the individual data items: EMPLOYEE_NAME, BADGE_NUMBER, SOC_SEC_NUMBER, and LOCATION. The generic form of any particular record is a *record type*. The relationship between different record types is defined by a *set*.

VAX DBMS uses *data definition languages (DDLs)* to define data and other database constructs. VAX DBMS has four DDLs: the schema DDL, the subschema DDL, the storage schema DDL, and the security schema DDL.

A *schema* is the logical description of the records and relationships between records (*sets*) making up a database. In effect, a schema is the overall logical definition of a database.

A *subschema* describes a subset (*user view*) of a database.

A *storage schema* describes the physical storage of records and sets of records in a database.

A *security schema* describes the access constraints on particular data structures.

Once you have defined a schema, you must compile it and load it into the *VAX Common Data Dictionary (CDD)*. The VAX CDD is the central VAX/VMS storage facility for information about data elements, data structures, and relationships between data structures for VAX DBMS and for VAX DATATRIEVE. The CDD does not contain the data itself. Storing a schema in this central facility allows the data definitions specified by the schema to be independent of the physical implementation of these definitions. Therefore, they do not need to appear in the application programs that use them.

You use the DDL compiler to load a schema into the CDD. When you load a schema into the CDD, the compiler automatically creates a default storage schema, a default security schema, and a default subschema. These are also stored in the CDD.

Creating a Database

Once a schema and its default storage and security schemas are stored in the CDD, you can create a database for that schema. You do this with the VAX DBMS operator (DBO) facility. Using the DBO/CREATE command, you generate a root file, which contains information about the structure of the database, and a set of data storage files. These files are stored as VAX/VMS files either in your default directory or in a directory of your choice. *The root file and the data files constitute the database.*

Using VAX DBMS, you can create any number of separate databases.

Retrieving and Storing Data with Database Query (DBQ)

You retrieve data in a VAX DBMS database by using *DBQ*, an interactive *data manipulation language (DML)* utility. (It also provides a callable interface to provide access from high-level languages.) If you run interactive DBQ on a VT100- or VT200-family terminal, DBQ uses a split screen to show the results of each DML statement you execute.

You use DBQ to retrieve data interactively as follows:

- Use the BIND command to access a particular database and specify the user view of that database you desire.
- Use the READY command to prepare the database (or selected parts of it) for your use.
- Use the FIND command to locate a particular record.
- Use the GET command to display the data of the currently located record on your screen.
- Use the FETCH command to perform the functions of FIND and GET on a particular record.
- Use a COMMIT command to make permanent the changes you have made; and a ROLLBACK command to erase any changes you have made.

The following illustrates data retrieval using DBQ:

```
dbq> BIND PARTS(RET)
dbq> READY(RET)
dbq> FIND FIRST PART(RET)
dbq> FIND OWNER WITHIN VENDOR_SUPPLY(RET)
dbq> GET(RET)
VEND_ID = 78912312
VEND_NAME = SANDEMAN TERMINALS SUBSYSTEMS
VEND_CONTACT = LABELLE M.K.
VEND_ADDRESS (1) = 57 GARRISON LANE
VEND_ADDRESS (2) = MADBURY, NH
VEND_ADDRESS (3) = 03280
VEND_PHONE = 6032478700
```

You use DBQ to store data interactively as follows:

- Use the BIND command to access a particular database and specify the user view of that database you desire.
- Use the READY command with the UPDATE qualifier to prepare the target user view records for updating.
- Use the STORE command to add a new record to the database.
- Use the COMMIT command to make the changes permanent; and a ROLLBACK command to erase any changes you have made.

The following illustrates storing new records in a database:

```
dbq> BIND PARTS(RET)
dbq> READY UPDATE(RET)
dbq> FIND FIRST CLASS USING CLASS_CODE(RET)
CLASS_CODE [X(2)] = ER(RET)
dbq> STORE PART(RET)
PART_ID [X(8)] = ER223456(RET)
PART_DESC [X(50)] = A DEMONSTRATION PART(RET)
PART_STATUS [X(1)] = G(RET)
PART_PRICE [S9(9,-3)] = 1.33(RET)
PART_COST [S9(9,-3)] = .98(RET)
PART_SUPPORT [X(2)] = EG(RET)
dbq> COMMIT(RET)
dbq>
```

Ensuring Consistency and Accuracy

To ensure data consistency and accuracy during use of a database by two or more users or programs, VAX DBMS suppports the principles of *transactions* and *record locking*.

A *transaction* is a series of operations that must execute as a complete unit or not at all. A transaction begins when a READY statement is executed and ends when a COMMIT or a ROLLBACK statement is executed. COMMIT causes all changes to the database (additions, deletions, modifications) to be entered in the physical database.

If any operation during a transaction is not completed successfully (for example, a payout is greater than a prescribed upper limit), you can cause a ROLLBACK statement to be issued. The ROLLBACK statement nullifies all operations made during the current transaction and terminates the transaction.

The principle of a transaction requires that a series of operations on a database are treated as a unit that must never be only partially completed. For example, you can use it to ensure that a new address cannot be entered in a database without a valid Zip Code.

A ROLLBACK is automatically issued if a transaction is interrupted for any reason, deliberate or otherwise.

VAX DBMS allows many users to read or write to a database concurrently. However, this feature introduces the possibility of inconsistency. For example, if one user is accessing a record to modify it while another user is accessing the same record to read it, the one reading it may read obsolete values.

VAX DBMS guards against inconsistency in a database by means of automatic *record locking*. When a particular record is accessed by a user or a program during a transaction, that record cannot be modified by another user or program until it is released by the first user or program.

Fine-Tuning a Database

Once you have tested the logical design of a database by using DBQ to retrieve data, you can edit the default storage schema to optimize certain storage or retrieval paths. You can also edit the default subschema, or create any number of new subschemas, to allow access to any desired selection of record types in the database instead of the entire database.

To edit an existing storage schema or subschema you use the DBO/EXTRACT command with the OUTPUT option. This command retrieves the schema source code from the CDD, and the OUTPUT option assigns the source code to a file. You can then edit the source code with your text editor.

When you are finished editing the source code, you use the DDL/COMPILE command to compile the altered storage schema or subschema and place it under the existing database schema in the CDD.

To create additional storage schemas or subschemas, you simply edit default schemas extracted from the CDD or create new source files from scratch. Then use the DDL/COMPILE command to compile the files.

By using the /SUBSCHEMAS, /STORAGE_SCHEMAS, and /ROOT options with the DBO/CREATE command discussed earlier, you can derive multiple databases from the same basic schema.

Accessing a Database From an Application Program

You can access a VAX DBMS database from any application program written in a high-level language that conforms to the VAX/VMS calling standard. These languages include:

VAX COBOL	VAX C
VAX DIBOL	VAX MACRO
VAX FORTRAN	VAX PASCAL
VAX BASIC	VAX PL/I
VAX BLISS-32	

For all languages that conform to the VAX/VMS calling standard, the interface to VAX DBMS is through callable DBQ. Callable DBQ is an interpreter to which you pass DBQ statements. If you use either COBOL or FORTRAN, you can embed data manipulation language statements directly in a program instead of calling DBQ.

To use the call interface, you must set up a *user work area (UWA)*. All data transfers between a program and a database take place within a UWA. COBOL and FORTRAN create the UWA for you; for other languages, you create a UWA with the DBO/WORK_AREA command.

When an application program is compiled, the included DML statements or the calls to DBQ automatically access the data definitions the program requires.

Using Database Operator (DBO) Utilities

The Database Operator provides you with utilities that allow you to:

- Determine what schemas, storage schemas, and subschemas are in a particular CDD directory (DBO/REPORT).
- Coordinate the deletion of databases, schemas, subschemas, and storage schemas from the CDD (DBO/DELETE).
- Get information on schemas, subschemas, and storage schemas from database root files (DBO/DUMP).
- Change the parameters chosen when a database was created or previously modified (DBO/MODIFY).
- Monitor the current use and status of a database (DBO/SHOW).

To invoke these utilities, you simply enter the command with the desired options. For example, to delete a database with the same name as its root file, ABSTRACT, you would enter:

```
$ DBO/DELETE/SCHEMA ABSTRACT
```

Protecting Database Security and Integrity

VAX DBMS provides several means of protecting the integrity and security of databases. *Integrity*, in this context, refers to the accuracy, completeness, and timeliness of a database; *security* refers to the ability to limit access to a database.

VAX DBMS provides database integrity by allowing you to:

- Use the schema CHECK clause to verify data at the time of storage or modification. The CHECK clause verifies the reasonableness of data values.
- Use DBO/VERIFY to ensure the syntactic correctness of records and the record sets in which they participate.
- Maintain backup copies of all databases.
- Enable long-term after-image journaling when you create the database or modify a schema.
- Monitor a database for usage, response, and potential reorganization.
- Correct data errors immediately and take action to correct the cause. Data errors can be application-dependent (a user enters inconsistent data) or system-dependent (an internal hardware or software error occurs).
- Run application programs against test databases to ensure that the programs work properly and do not alter the database.

VAX DBMS provides database security by allowing you to:

- Define security schemas to limit data access
- Segment a database into areas and then assign any desired VMS file protection to the files containing those areas
- Assign particular areas to separate storage devices that can then be taken off line whenever desirable
- Limit record access through the use of tailored subschemas
- Use CDD facilities to restrict access to data definitions stored in the CDD

Using VAX DBMS with Other Products

VAX DBMS interacts directly with two other products in the VAX Information Architecture family.

The VAX Common Data Dictionary (CDD)

VAX DBMS uses the CDD as a central storage facility for the logical definition of a database (schema), the specification of how the database is physically stored on mass storage devices (the storage schema), individual user views of the database (subschema), and the access constraints assigned to various database entities (security schema). The CDD does not contain any data. The data that VAX DBMS accesses is physically stored in VAX/VMS files; these files are pointed to by information contained in the CDD.

The CDD must be available to VAX DBMS. It has its own dictionary management utility, called DMU, which provides extensive facilities for examining and maintaining CDD contents.

VAX DATATRIEVE

The VAX DATATRIEVE interface to VAX DBMS lets you retrieve, store, report, and manipulate data in database files created and managed by VAX DBMS. Using optional clauses in VAX DATATRIEVE *record selection expressions*, you can readily work with the complex data relationships that characterize CODASYL-type databases.

The following VAX DATATRIEVE verbs are reserved for use with VAX DBMS databases: CONNNECT, DISCONNECT, RECONNECT, COMMIT, and ROLLBACK.

Combined with DECnet, VAX DATATRIEVE provides an efficient means of accessing VAX DBMS databases at remote locations.

VAX DBMS does not require that VAX DATATRIEVE be available to it.


VT220

Chapter 6 ▪ VAX Rdb/VMS

VAX Rdb/VMS is a general purpose *relational* database management system. It provides the benefits of a full-function database management system — including data independence, data integrity and security, centralized administration, and the capability of handling concurrent access by multiple users — plus additional advantages to be discussed shortly below.

There are currently two VAX/Rdb products: VAX Rdb/VMS and VAX Rdb/ELN. VAX Rdb/VMS runs on the VAX/VMS and MicroVMS operating systems; VAX Rdb/ELN runs under the VAXELN operating environment. Beyond this basic difference, the two products are very similar.

With VAX Rdb/VMS you can:

- Create and maintain relational databases
- Store, retrieve, update, and delete data
- Separate data definitions from the application programs that use them
- Store all data definitions in the database files, and optionally in the VAX Common Data Dictionary (CDD), for common use and maintenance
- Establish relationships between records dynamically
- Tailor user views of data
- Centralize the administration of data
- Maintain data integrity and security
- Use databases concurrently with other users
- Ensure against data inconsistency during concurrent updating
- Recover from hardware and system failures

▪ Who Uses VAX Rdb/VMS

The choice of a data management product depends on many factors, including the size and complexity of the database (data items and relationships); the storage and user capacity of the system; the number of concurrent users; and the types of operations they perform. The most important considerations when choosing a database management system are the suitability of the data model for the particular application and the environment in which the application is developed.

In general, VAX Rdb/VMS is intended for use in applications that meet the following criteria:

- The database must be easy to understand and use.

  Because a relational database organizes data into tables, even people without knowledge of database management can understand the structure of a database.

- The structure of the database is likely to need frequent changing.

  VAX Rdb/VMS facilitates easy, interactive restructuring of a database. As the needs of your organization change you can add or change indexes, fields, views, relations, and protection parameters. Similarly, you can delete outdated information. You can also build prototype systems to test the structure of a VAX Rdb/VMS database without committing extensive resources.

- Data relationships can be defined at any time.

  Because VAX Rdb/VMS forms relationships between data on the basis of values stored in the database, not on predefined data structures, you can define relationships between data dynamically during a database session.

- A professional database administrator need not be available to create and maintain the database.

  A programmer can set up a VAX Rdb/VMS database using a simple set of statements, typed interactively at the terminal or entered in a command file, to translate a logical database design into a working database. Database maintenance is just as easy — a programmer or system manager can maintain the database.

- The data and information to be included in the database is unformatted.

  VAX Rdb/VMS supports unformatted data types, also called "segmented strings," that allow the database to be used to collect and access voice transmissions, graphics, word processing documents, laboratory data, satellite transmissions, and other uncommon data.

- The application requires remote database access and update capabilities.

  In situations where the data is distributed among various different VAX systems, VAX Rdb/VMS supports remote database access. This enhances the user's ability to collect data at remote sites, yet interact with other databases on other systems.

Because the relational model is easy to understand, VAX Rdb/VMS is useful for quick implementation of simple applications. VAX Rdb/VMS is also sophisticated enough to handle some relatively complex database applications.

▪ Benefits of VAX Rdb/VMS

- As a layered VMS product, VAX Rdb/VMS runs on the full line of VAX processors, allowing you to transport Rdb/VMS applications from one system to the next as your needs dictate.
- A single utility, the relational database operator facility (RDO), handles all database operations, including data definition and data manipulation. This allows you to create or change relations, user views, and integrity and security constraints. The RDO also lets you interactively retrieve and store data.
- Through the use of data structures called *relations*, relationships between data are established dynamically.
- Callable RDO and embedded data manipulation (DML) statements allow you to access VAX Rdb/VMS databases from application programs. Precompilers are provided for embedded DML statements in programs written in the following VAX high-level programming languages — VAX BASIC, VAX COBOL, VAX FORTRAN, and VAX PASCAL. A high-level call interface allows you to call VAX Rdb/VMS from any language that supports the VAX calling standard.
- A READ ONLY (also called "snapshot") mode allows read-only transactions to run independently of other users' jobs.
- A single process can access more than one VAX Rdb/VMS database simultaneously.
- Security provisions for data, data definitions, data manipulation statements, and database operator commands protect you from undesired access to VAX Rdb/VMS databases.
- Several users can access the same VAX Rdb/VMS database concurrently for storage, retrieval, update, and deletion.
- Support for *transactions* and *record-locking* to ensure data consistency and integrity during concurrent access.
- Before- and after-image journaling to ensure the integrity of the database in the event of hardware or software failures.
- VAX DATATRIEVE provides interactive access to a VAX Rdb/VMS database for ad-hos query and update.
- VAX Rdb/VMS interacts with several other VAX Information Architecture products.

- Using DECnet, VAX Rdb/VMS can access and update VAX Rdb databases on remote VAX/VMS and VAXELN nodes.
- VAX Rdb/VMS includes facilities for database backup and restoration.
- The structure of a VAX Rdb/VMS database is easy to understand and easy to explain to users.
- Using VAX Rdb/VMS, programmers can develop applications more quickly than with conventional third generation programming techniques.
- A VAX Rdb/VMS database allows you to combine and compare data in a variety of ways.
- A highly trained database administrator is usually not required. A programmer or an analyst can create, modify, and maintain a VAX Rdb/VMS database.

▪ A Closer Look at VAX Rdb/VMS

Using VAX Rdb/VMS involves:

- Designing the database in accordance with your information needs
- Creating the database, defining its elements
- Restructuring the database
- Deleting database elements
- Accessing and retrieving data
- Modifying data
- Ensuring consistency and accuracy
- Maintaining the database
- Accessing VAX Rdb/VMS from application programs

Designing a Database

Designing a relational database involves analyzing the data flow within your organization and then grouping related data items into table-like structures called *relations*. A logical relation can be visualized as a physical table of related data. For example, a relation named EMPLOY might be visualized as follows in Figure 6-1:

FIRST_NAME	LAST_NAME	EMPLOYEE_ID	SEX	DEPARTMENT_CODE
George	Toliver	842	M	ENG
Paul	Blanchett	217	M	MKTG
Christine	Decker	919	F	RECR
Paul	Dallas	426	M	MNFG
James	Cool	119	M	ADMN
Russ	Stuart	554	M	ENG
Leslie	Neil	721	M	ADMN
Nancy	Brown	842	F	PERS
George	Boutin	740	M	ENG
Ann	Patterson	328	F	SALE

Figure 6-1 ▪ Sample Rdb/VMS Relation

A row in a relation is called a *record*. The individual components of a row are referred to as *fields* — a field can contain only a single item of data. In the example, a record consists of the fields FIRST_NAME, LAST_NAME, EMPLOYEE_ID, SEX, and DEPARTMENT_CODE. A field that uniquely identifies a record in a relation can be used as a *key field* for accessing that record. In the above example, the field EMPLOYEE_ID might serve as a key field because no two records can contain the same employee ID number.

By placing a common field in separate relations you can link one relation with another. For example, by placing the field EMPLOYEE_ID in a relation containing employee information (name, address, home phone, emergency contact, etc.) and the same field in a separate relation containing job history information, you can then pull any desired data from these two relations and combine the data to form a new, temporary relation that fulfills a specific informational need. Linking together two or more relations is called a *join* operation.

The process of defining a complete set of relations that models the data needs of an organization involves:

- Determining the data needed
- Grouping data into categories on the basis of relatedness
- Eliminating redundancy by finding items occurring in more than one place and putting them in a single location
- Moving repeating items in a field to separate relations (a field can contain only a single item)
- Determining which field or fields in a relation are to serve as an index for accessing records in the relation
- Including a common field in relations so that dynamic relationships may later be established
- Dividing the database into views, for efficiency and clarity of structure
- Defining constraints for each relation, for data integrity
- Defining access privileges for the database and for individual relations

Creating a Database and Defining its Elements

You use the relational database operator utility (RDO) to create a database. RDO allows you to define a database either by entering DEFINE statements after the RDO prompt (RDO>), or by using your text editor to enter DEFINE statements as an indirect command file.

The DEFINE statements you use to define the logical structure of a VAX Rdb/VMS database are:

- DEFINE DATABASE

  Gives the database a *name* and associates this name with a *path name* in the VAX Common Data Dictionary (if available). The path name specifies the dictionary directory in which the database definitions are to be stored. If the CDD is not installed on your system, VAX Rdb/VMS stores the database definitions only in the database file.

  In addition to naming the database, the DEFINE DATABASE statement allows you to name the database file and the *snapshot* file. A snapshot allows you to retrieve data without interfering with other users or risking record locks. If you do not name these files explicitly, VAX Rdb/VMS assigns them the same name as the database and assigns them default device, directory, and file types. Optional clauses let you determine how the database uses space in memory.

The following defines a database named OVERNITE:

```
DEFINE DATABASE OVERNITE IN CDD$TOP.ACCOUNTING.
     OVERNITE FILE IS DISK2:[ACCOUNTING]OVERNITE.RDS.
```

- DEFINE FIELD

  Adds a field definition to the database file and to the CDD. Once you have defined a field with a DEFINE FIELD statement, you can include it in any relation definition simply by naming it. You can also define a field without using a DEFINE FIELD statement. You do this within a relation definition by giving the field a name and specifying its characteristics.

  The following defines a field named TOTAL_CHARGE having a *data type* of F_FLOATING:

```
DEFINE FIELD TOTAL_CHARGE DATATYPE IS F_FLOATING.
```

  The data type of a field constrains the value of that field to the range of values defined by the data type. The F_FLOATING data type, for example, includes only single-precision floating-point numbers (to approximately seven decimal places).

- DEFINE RELATION

  Creates a relation definition. A relation definition consists of a list of field definitions. The simplest way to define a relation is to list the names of fields that have been previously defined in DEFINE FIELD statements. You can also define a relation by defining within the relation itself the fields that are to compose the relation, that is, by giving each field a name and specifying its characteristics.

  The following defines a relation named RESERVATION:

```
DEFINE RELATION RESERVATION.
  ROOM_NUMBER.
  TX_DATE.
  GUEST_NAME.
  ARRIVAL_DATE.
  DEPARTURE_DATE.
END RESERVATION RELATION.
```

- DEFINE VIEW

  Creates a *view* definition. A view is a "virtual relation" that includes field definitions from one or more separate relations. The fields and records of a view are not stored physically, but are simply pointed to by the descriptions in the view definition.

  Using views saves you from using the same query over and over. The following example defines a view named ROOMS:

```
DEFINE VIEW ROOMS OF H IN HOTEL
  CROSS T IN TYPE OVER ROOM_TYPE
  CROSS R IN RATES OVER RATE_CODE.
     NUMBER    FROM    H.ROOM_NUMBER.
     TYPE      FROM    H.ROOM_TYPE.
     TEL       FROM    T.BEDS.
     TUBE      FROM    T.TELEPHONE.
     AIR       FROM    T.AC.
     RATE      FROM    R.RATE_CODE.
     GOV       FROM    R.GOV_RATE.
     GROUP     FROM    R.GROUP_RATE.
     STD       FROM    R.STD_RATE.
```

  This view defines a virtual relation named ROOMS and gives new, more convenient names to existing fields in the component relations, TYPE and RATES.

- DEFINE CONSTRAINT

  Creates a constraint definition. A constraint definition defines the set of conditions that restrict the values in a field. You can also define constraints by placing VALID IF clauses in DEFINE FIELD statements.

  The following example defines a constraint named RATE_CODE_EXISTS:

```
DEFINE CONSTRAINT RATE_CODE_EXISTS
  FOR H IN HOTEL
    REQUIRE ANY R IN RATES
    WITH R.RATE_CODE = H.RATE_CODE.
```

  In this example, if you attempt to store a record in HOTEL that contains a RATE_CODE value that does not match an existing code in the RATES relation, VAX Rdb/VMS returns an error; it does not store the record.

- DEFINE INDEX

  Creates a single or multisegment *index-key* definition for a relation. An index key allows you to locate a record on the basis of the value in a particular field.

The following defines an index named HOTEL_ROOM_NUMBER:

```
DEFINE INDEX HOTEL_ROOM_NUMBER FOR HOTEL
  DUPLICATES ARE NOT ALLOWED,
    ROOM_NUMBER,
END HOTEL_ROOM_NUMBER INDEX.
```

This example makes the field ROOM_NUMBER, contained in the OVERNITE relation mentioned earlier, an index; the name is useful only when deleting the index.

- DEFINE PROTECTION

  Creates an entry within an *access control list (ACL)* for a database or relation. An ACL entry contains a user identifier and a list of access rights granted to that user. VAX Rdb/VMS stores the ACL for a database — and for each of its relations — in the database file.

  The following example specifies the position of the access control list entry in the access control list, the identifier of the user(s) to whom the entry applies, and the list of access rights to be granted or denied to the user(s):

```
DEFINE PROTECTION FOR DATABASE OVERNITE
  POSITION 6
  IDENTIFIER [00240,*],
  ACCESS NOOWNER+NOOPERATOR+NOADMINISTRATOR,
```

Restructuring a Database

You use CHANGE statements to restructure a database by making changes to existing definitions. The CHANGE statements can change definitions in the CDD as well as in the database file.

The CHANGE statements change definitions, not data. When a user accesses data that was created with a definition before that definition was changed, VAX Rdb/VMS converts the data to conform to the new definition automatically.

- CHANGE DATABASE

  Changes the definition for an existing database. You can change the name of the database file or journal file, the CDD path name, the size of the database file, and buffer parameters.

- CHANGE FIELD

  Changes a field definition.

- CHANGE RELATION
  Changes a relation definition. Within this statement you can add, delete, or change the definition of any of the relation's fields.

- CHANGE PROTECTION
  Changes the access rights for an entry in an access control list.

Deleting Database Elements

You use the DELETE statement to remove the definition of a database component from the database file and from the CDD.

The DELETE statements are as follows:

- DELETE DATABASE
  Deletes entire database

- DELETE FIELD
  Deletes a field definition

- DELETE CONSTRAINT
  Deletes one or more constraint definitions

- DELETE INDEX
  Deletes one or more relation index definitions

- DELETE PROTECTION
  Deletes an entry from the access control list for a database, relation, or view

- DELETE RELATION
  Deletes one or more relation definitions

- DELETE VIEW
  Deletes one or more view definitions

Retrieving Data

You can retrieve data from an Rdb/VMS database in one of several ways:

- *Through application programs* — most often, you will retrieve data by using VAX Rdb/VMS statements included directly in high-level language programs.
- *Through VAX DATATRIEVE* — interactive queries, simple reports and graphics can be handled most easily by DATATRIEVE.
- Through *RDO, the interactive utility* — RDO provides a tool for prototyping, testing and learning. A statement that works in RDO will work the same way when included in a program.

Retrieving data from an Rdb/VMS database involves:

- Using a FOR or START_STREAM command, modified with a *record-selection expression*, to form a temporary grouping of records called a *record stream*.

  The record stream can comprise any desired selection of records, whether from a single relation or from several relations joined together.
- Using a PRINT or GET command to select the fields from the records. The PRINT statement displays values on the terminal. The GET statement assigns database values to variables in a program.

In most cases, you would use a FOR loop (containing a FOR statement and a PRINT statement) to form a record stream and display selected fields. The following is an example of a FOR loop:

```
                record selection expression
                          ¦
      +-------------------+-----------------------+
FOR ¦ E IN EMPLOYEES WITH E.DEPARTMENT = "ENG" ¦
      +-----------------------------------------+
        PRINT E.LAST_NAME, E.FIRST_NAME
END_FOR
```

The record selection expression in this example begins with the letter E, which is called the *context variable*. The context variable is the name you use to associate a variable with a particular relation and, therefore, the individual fields (for example, E.LAST_NAME) in the relation records.

The result of the FOR statement in this example, illustrated by Figure 6-2, is a record stream consisting of all the EMPLOYEES records with the string "ENG" in the DEPARTMENT_CODE field.

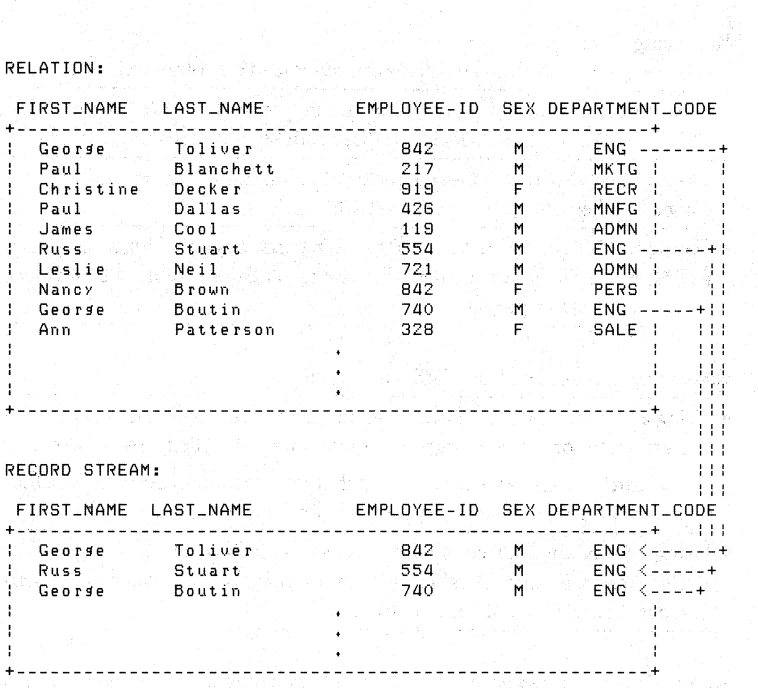


Figure 6-2 ▪ Result of Example FOR Loop

The PRINT statement in this example selects only the fields E.LAST_NAME and E.FIRST_NAME for display on your terminal.

Using a combination of RSE clauses and *value* and *conditional expressions* you can limit the record stream formed by a FOR statement in many different ways.

The RSE clauses you can use are as follows:

- FIRST n
  Retrieves only the number of records specified.

- WITH
  Names a set of criteria for selection, using conditional expressions.

Example:

```
FOR E IN EMPLOYEES WITH E.LAST_NAME = "Toliver"
     PRINT
     E.FIRST_NAME,
     E.LAST_NAME,
     E.SOCIAL_SECURITY
END_FOR
```

- CROSS
  Names a second relation for a join operation.
  Example:

```
FOR E IN EMPLOYEES
     CROSS D IN DUMPS
     OVER CODE
END_FOR
```

- SORTED BY
  Names a key with which to sort the record stream.
  Example:

```
FOR E IN EMPLOYEES SORTED BY E.STATE
   PRINT
   E.STATE,
   E.TOWN,
   E.EMPLOYEE_ID
END_FOR
```

- REDUCED TO
  Names a field to serve as the reduced to key. The record stream consists of the unique values for that field. This is sometimes called a *project* operation.
  Example:

```
FOR E IN EMPLOYEES REDUCED TO E.TOWN
   PRINT E.TOWN
END_FOR
```

VAX Rdb/VMS permits the full range of selection criteria and operators to retrieve exactly the desired records and perform standard operations on them.

- Conditional (or Boolean) expressions — These expressions perform range retrieval and pattern matching, and check for existence and uniqueness. Examples:

The most common type of *value expression* is the simple field name (for example, E.LAST_NAME); others include:

Value Expression	Example
Literals	"ENG"
Arithmetic	E.SALARY_AMOUNT * 1.3
Statistical	AVERAGE E.DEPARTMENT_CODE OF E IN EMPLOYEES WITH E.DEPARTMENT_CODE = "ENG"

A *conditional expression* compares two value expressions and evaluates to either TRUE or FALSE. The conditional expression operators are:

EQ =

NE <>

GT >

GE >=

LT <

LE <=

You can use the logical operators AND, OR, and NOT to combine conditional expressions.

The following illustrates a multiconditional FOR loop:

```
FOR E IN EMPLOYEES WITH E.STATE = "Massachusetts"
  CROSS D IN DEPARTMENTS OVER DEPARTMENT_CODE
  AND (D.DEPARTMENT_NAME = "Engineering"
  OR D.DEPARTMENT_NAME = "Manufacturing")
    PRINT E.LAST_NAME,
       E.FIRST_NAME,
    END_PRINT
END_FOR
```

You can also use value and conditional expressions in PRINT statements. For example:

```
FOR E IN EMPLOYEES
      +-----------------------+
PRINT ¦ E.SALARY_AMOUNT * 1.3 ¦ END_FOR
      +-----------+-----------+
                  ¦
        arithmetic expression
```

A FOR statement works well when you want to process the records from a single record stream one at a time. There may also be times when you want to establish more than one record stream, and you want the processing of the streams to interact. In such cases, use a START_STREAM statement to start each stream.

Think of the START_STREAM statement as analogous to the OPEN statement in a high-level programming language like BASIC. After you use OPEN to open a file in BASIC you use GET statements to put records into a buffer for processing, one record at a time. Similarly, in VAX Rdb/VMS, after you set up a record stream with the START_STREAM statement, you use FETCH statements to make each successive record in the stream available for processing.

This START_STREAM/FETCH combination differs from the use of a FOR loop containing a GET or PRINT statement. When you use a FOR loop, VAX Rdb/VMS automatically performs a GET or PRINT once for each record in the record stream set up by the RSE. With START_STREAM, the RSE sets up a record stream, but VAX Rdb/VMS does not set up a loop and automatically process each record in the record stream. Instead, you must issue a FETCH statement to advance a record pointer to each succeeding record in the stream. After you have advanced the pointer, you can then use data manipulation statements to operate on the specific record.

The FOR loop is easier to use than START_STREAM; however, START_STREAM gives your program more flexibility. For example, you can start more than one record stream, and the values returned from one stream can affect the processing of the other.

Modifying Data

Rdb/VMS allows you to store, modify, and erase data using the same kind of record selection expression you use to retrieve. With it you can precisely specify the records you want to change.

- STORE

  Stores values in the database. For example, the following statement stores a new DEPARTMENTS record:

```
STORE D IN DEPARTMENTS USING
      D.CODE = "RECR";
      D.NAME = "Recreation";
      D.EMPLOYEE_ID = 567;
END_STORE
```

- MODIFY
  Modifies values in the database. For example, the following statement changes the manager of the Recreation Department:

```
FOR D IN DEPARTMENTS WITH D.CODE = "RECR"
   MODIFY USING
      D.EMPLOYEE_ID = 433
   END_MODIFY
END_FOR
```

- ERASE
  Deletes records from relations. A budget cut has spelled the end of the Recreation Department:

```
FOR D IN DEPARTMENTS WITH D.CODE = "RECR"
   ERASE D
END_FOR
```

Ensuring Consistency and Accuracy

To ensure data consistency and accuracy during use of a database by two or more users or programs, VAX Rdb/VMS supports *transactions* and *record locking*.

A *transaction* is a series of operations that must execute as a unit or not at all. A transaction begins when a START_TRANSACTION statement is issued, and ends when a COMMIT or ROLLBACK statement is issued. COMMIT causes all changes made during a transaction to be entered in the physical database.

If any operation during a transaction is not completed successfully (for example, a ZIP Code validation operation fails), the user may issue a ROLLBACK statement. The ROLLBACK statement nullifies all operations in the current transaction.

VAX Rdb/VMS lets many users read, write, and modify data in a database concurrently. This feature also introduces the possibility of inconsistency. For example, if two users read and then modify a single field, one of them may be reading an obsolete value. In general, a program updating a value must be sure that the update will be completed before any other program can update the same value.

VAX Rdb/VMS ensures this kind of consistency by means of automatic record locking, and by allowing your program to explicitly protect relations and records from actions by other programs during a transaction.

Through automatic record locking, when a record is being accessed by one program or user during a transaction, that record cannot be modified by another user until the current transaction is completed and either the record is modified or simply released.

When you issue the START_TRANSACTION statement, you can include a list of relations and the kind of access to be allowed to other programs during your transaction. For example, if your program is updating the EMPLOYEES relation, it might start a transaction with PROTECTED WRITE access as follows:

```
START_TRANSACTION RESERVING EMPLOYEES FOR PROTECTED WRITE
```

This statement allows other programs to access the EMPLOYEES relation for reading data in the relation, but not for writing data. Other programs cannot write data until your program completes the transaction with a COMMIT or ROLLBACK statement.

Maintaining a Database

The Rdb *RDO* facility provides a set of statements that allows you to perform common database maintenance functions, such as backing up, restoring, analyzing space usage, checking database integrity, and maintaining journal files to restore a database if there is a failure. These statements are as follows:

- ANALYZE

  The ANALYZE statement displays the space usage per page for the database file. Optional qualifiers display the number of records and the index structure for each relation within the database. Regular analysis of database usage lets you restructure your database to improve processing efficiency.

- BACKUP and RESTORE

  The BACKUP statement writes a compressed copy of the database to a file. If the database becomes corrupted through a hardware failure, or in some other way, you can use the RESTORE statement to install the saved ve rsion of the database.

- RECOVER

  In case of a system failure, you can use the RECOVER statement to apply a journal file to a backup copy of the database.

Accessing a VAX Rdb/VMS Database from Application Programs

VAX Rdb/VMS is optimized for use by application programs.

You can access a VAX Rdb/VMS database from an application program written in any high-level language that supports the VAX standard call interface. These languages include:

VAX COBOL	VAX C
VAX DIBOL	VAX MACRO
VAX FORTRAN	VAX PASCAL
VAX BASIC	VAX PL/I
VAX BLISS-32	VAX RPG II

You access a VAX Rdb/VMS database from an applications program:

- By using callable RDO (applies to all the languages listed above), or
- By using precompiled data manipulation statements (applies to COBOL, BASIC, FORTRAN, and PASCAL only)

▪ Using VAX Rdb/VMS with Other Software Products

In some cases, VAX Rdb/VMS uses other VAX Information Architecture products automatically. In other cases, you can use products together explicitly. Either way, the VAX Information Architecture ensures that the products you choose work together smoothly.

VAX Rdb/VMS and the Common Data Dictionary

You may store definitions of VAX Rdb/VMS database entities in the VAX Common Data Dictionary (CDD). For example, if you are working in the default dictionary directory called CDD$TOP.RDB and you create a database called PERSONNEL, VAX Rdb/VMS installs a directory in the CDD called CDD$TOP.RDB.PERSONNEL. Then, when you define fields, relations, and other elements of the database, VAX Rdb/VMS places these definitions in this directory.

When you store VAX Rdb/VMS definitions in the CDD, other VAX Information Architecture products and high-level language compilers and precompilers can access them. For example, if you wish to use VAX DATATRIEVE to make ad hoc queries to a VAX Rdb/VMS database, the VAX Rdb data definitions must be located in the CDD.

VAX Rdb/VMS and DATATRIEVE

You can use VAX DATATRIEVE and VAX Rdb/VMS together:

- To make ad hoc queries.

  An *ad hoc query* is a data retrieval operation you perform once to get a specific piece of information from the database. By using VAX DATATRIEVE as its interactive query facility, VAX Rdb/VMS takes advantage of many capabilities for information retrieval and manipulation not available with many relational database query languages. You can use VAX DATATRIEVE to query a VAX Rdb/VMS database without being concerned about how VAX DATATRIEVE accesses the data. VAX DATATRIEVE can also be used to access data stored in RMS files or in a VAX Rdb/VMS or VAX DBMS database.

- For prototyping applications.

  VAX DATATRIEVE also provides a simple tool for designing and testing an application. For example, programmers already familiar with VAX DATATRIEVE might prefer to use it to model the queries and reports that will be included in an application, rather than using RDO. Then these queries can be translated into VAX Rdb/VMS statements. The VAX Rdb statements that retrieve, modify, and erase data closely resemble VAX DATATRIEVE, so the translation process is easy.

- For developing simple applications.

  You can write VAX DATATRIEVE applications that access VAX Rdb/VMS databases. VAX DATATRIEVE is especially useful for applications in which:

  - Your application performs elaborate calculations or transformations of data.
  - Your application produces a complex report whose form is important.
  - Your application calls for graphs and tables included in its output.

  For such applications, it is easier to write VAX DATATRIEVE procedures than high-level language programs.

VAX Rdb/VMS and Other VAX Information Architecture Products

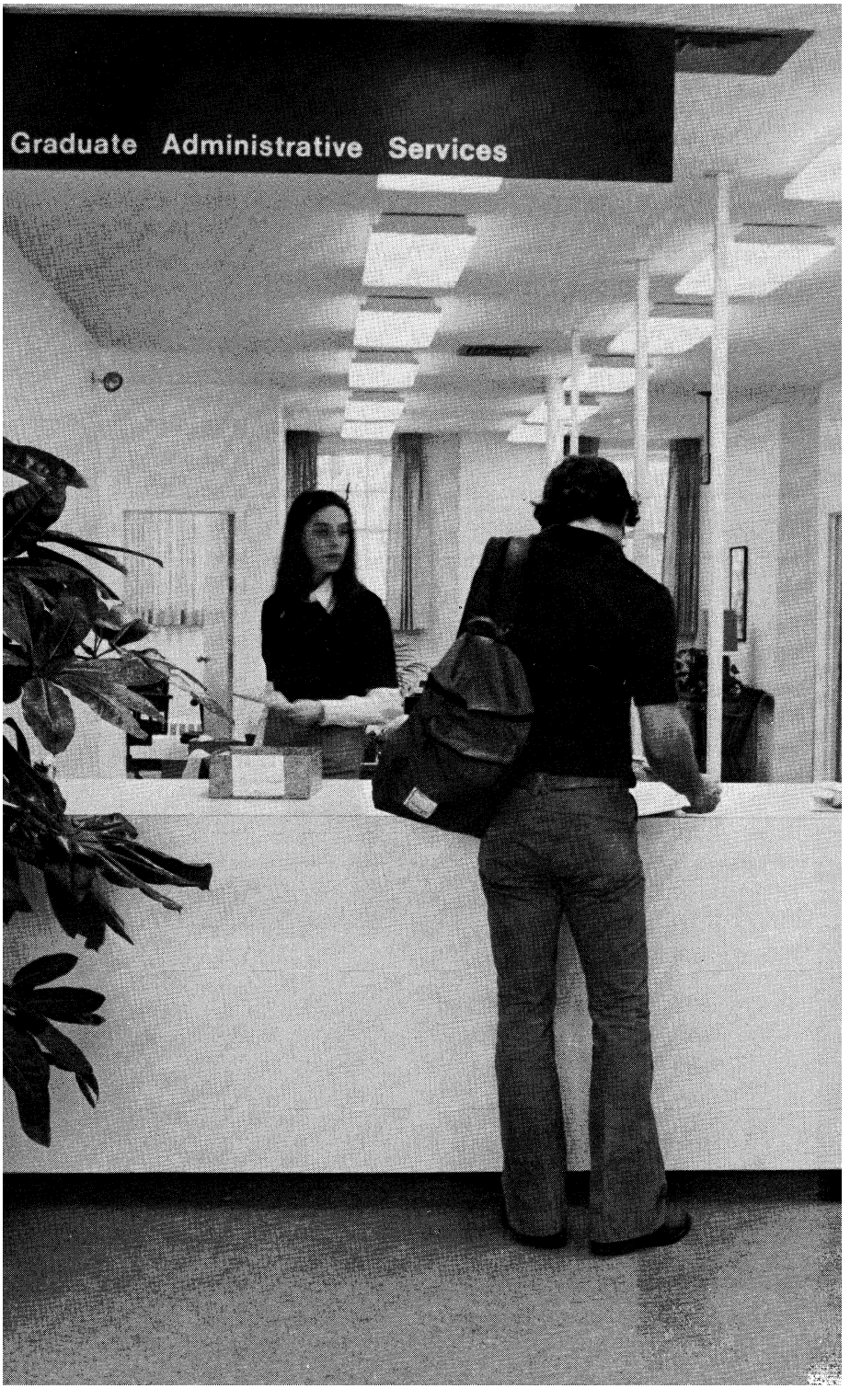
VAX Rdb/VMS can be integrated with the other VAX Information Architecture products as well:

- VAX ACMS
  To develop and manage transaction processing and other complex applications that access data using VAX Rdb/VMS.

- VAX TDMS
  To define forms and manage terminal I/O in applications that store data in and retrieve data from VAX Rdb/VMS databases.

- PROGRAMMING LANGUAGES
  To write application programs that access a VAX Rdb/VMS database using any VAX languages which support the VAX calling standard.

- VAX DECgraph
  To generate graphs and plots of data accessed through VAX DATATRIEVE.


Graduate Administrative Services

Chapter 7 • VAX TDMS and VAX FMS

Digital offers two alternative solutions — VAX TDMS and VAX FMS — in the area generally referred to as terminal management for VAX systems. TDMS and FMS are alike in many ways, but also have some significant differences which will be discussed at the end of this chapter. VAX users have a choice, then, and it is highly likely that either TDMS or FMS can provide an excellent terminal management solution for their particular needs.

This chapter will first discuss the VAX Terminal Data Management System (TDMS), then the VAX Forms Management System (FMS), each being viewed as an independent product. The two will then be compared and contrasted, with advice offered as to which product is most appropriate in which situation(s).

VAX TDMS

VAX TDMS, Digital's Terminal Data Management System, is a programmer productivity tool designed to reduce the high lifecycle costs of developing and maintaining forms-intensive terminal applications on VMS systems.

TDMS offers a wide range of features making it easy to develop applications that display and collect information, relieving the programmer of many of the "burdens" associated with conventional forms-based applications. TDMS is an integrated member of the VAX Information Architecture, as it uses the VAX Common Data Dictionary and can also interact with VAX DATATRIEVE, VAX DBMS, VAX Rdb/VMS, and VAX ACMS. It provides a record-level interface, while VAX FMS offers a field-level interface.

TDMS provides a fourth-generation interface for defining the data exchange between screen(s) and program(s). It is used in conjunction with standard third-generation languages such as VAX COBOL, VAX BASIC or VAX FORTRAN for defining programming logic. It replaces the often cumbersome coding of program/screen interaction with definitions, which are stored independently in the VAX CDD.

Use of a record-level interface provides both terminal and data independence at the application program level, as the application need only exchange record buffers with the interface; as simple as programming to a disk file. TDMS manages the moving, mapping, and data type conversion between the fill record data types and screen data types, freeing the application of all such logic.

With VAX TDMS you can:

- Define the exchange of data between an application program and its associated terminal(s) using a nonprocedural language. These predefined exchanges of data, called REQUESTS, are external to the application programs they serve.
- Use a screen editor for defining forms, allowing you to easily format data on your terminal. Forms are defined external to the application program and can often be modified without resulting in a corresponding modification to the application program that uses it.
- Use standard utilities enabling the creation, modification, and storage of FORM definitions and REQUEST definitions.
- Use a "record level" programming interface which the application program uses to invoke and execute the predefined REQUESTs. The application program calls the program interface, passing it the REQUEST name and the program record buffers used in the exchange of data with the terminal.
- Store form definitions and request definitions in the Common Data Dictionary along with the file record definitions that the requests reference. The CDD acts as a central repository for these definitions, allowing them to be shared among multiple applications. Changes are made once, and frequently can be made without affecting the application program that uses them.

▪ Who Uses VAX TDMS

Virtually anyone needing to reduce the high costs of developing and managing forms-intensive terminal applications on VAX systems can benefit from using VAX TDMS.

It is intended for use primarily in complex programming efforts, those requiring at least one man-year to complete. TDMS can provide benefits to application managers, application designers, application programmers, and terminal operators. It is also a prerequisite for all users of VAX ACMS.

The program in a TDMS application can be viewed as a generic algorithm that executes a series of requests (or routines) and reads and/or writes information to a database. The requests and form definitions are independent of the program and can thus be modified without incurring significant programming costs.

▪ Benefits of VAX TDMS

- TDMS uses nonprocedural definitions for forms, records, and requests. With nonprocedural definitions, users can define their terminal data management logic *external* to the application program. *Development and maintenance are simplified, and a better division of skills in your development team can be attained.*
- The screen editor used with TDMS defines field validation and completion attributes including character validation, display field characteristics, range checks, check digits, list checks, and screen-wide characteristics. Again, *productivity is improved by virtue of the fact that field validation is performed external to the application program.*
- TDMS features a record level interface, allowing the application program and appropriate form to exchange data a record at a time. *The field/mapping conversion is automatic, further reducing required programming effort.*
- TDMS offers *device independence.* That is, *application programmers do not need to concern themselves at all with the specific type of device a terminal operator will be using.* Terminal manipulation (such as cursor control, scrolling, and video highlighting) is defined by the form and the request, wholly independent of the application programming.
- In summary, TDMS *reduces the development and maintenance costs* associated with complex, terminal-based applications.

▪ A Closer Look at VAX TDMS

The primary goal of VAX TDMS is to significantly reduce programming costs during the life cycle of an application. TDMS lets you realize these savings by replacing major portions of the application program with definitions that are created and stored *outside* of the application program, including:

- Form Definitions, which define the image that appears on the terminal, as well as data input requirements
- Record Definitions, which define the datatype, structure, order, and length of the records that the application uses
- Requests, which define the exchange of information between the program (record definition) and the terminal (form definition), including data input and output

These definitions are all stored in one central repository — the VAX Common Data Dictionary. The CDD is used indirectly by all users of TDMS, and should changes to definitions be needed, those changes are only made in the CDD. Because these definitions are the output of utilities and not written as part of the program code, it is often possible to revise your application without changing the application program.

For example, suppose that you develop an application that uses forms for your personnel department. Six months after the application is installed and running, the personnel department tells you that employee identification numbers will change from five-digit to nine-digit numbers. If the application has been developed without TDMS, you incur a major cost of revising, debugging, and recompiling program code. With TDMS, however, the process is greatly simplified: the elements of the application that must be changed are kept outside the program. As a result, you may only need to revise the offline definitions; in many cases, you will not have to change one line of program code.

Every TDMS application includes the following parts, or elements:

- An application program
- One or more record definitions
- One or more form definitions
- One or more requests
- One or more request library definitions
- One or more request library files

The Application Program

The program in a TDMS application:

- Reads data from and writes data to the database
- Uses the TDMS programming calls to:
  - Execute requests
  - Open and close request library files
  - Open and close I/O paths to the terminal
  - Read or write text from the reserved message line on the terminal
  - Cancel a call in progress
  - Signal the return status for TDMS call errors
  - Activate a facility (TRACE) that traces the action of a request
- Executes the program logic
- Provides for error processing

TDMS also performs datatype conversion during the execution of a request, converting form text to the datatypes of receiving record fields on input and converting the datatypes of record fields to form text on output.

In short, the program identifies the requests that are to be executed. The request controls the flow of data between the record and the terminal, and the program controls the flow of data between the record and the disk.

For a TDMS application program, you can use any VAX native mode language that adheres to the VAX Procedure Calling and Condition Handling standard. Application programs written in VAX COBOL or VAX BASIC can additionally benefit from the CDD record extraction capabilities by eliminating the need to redefine records.

Record Definitions

VAX TDMS uses *record definitions* to identify the datatype, structure, and length of records used in an application. For application programs that are written in languages supporting the VAX Common Data Dictionary (CDD), you need only reference the record definition that already exists in the CDD rather than redefine the record in your program. VAX COBOL V2.0 (the COPY statement) and VAX BASIC V2.0 (the %INCLUDE statement) currently support the extraction of records from the Common Data Dictionary. (If your application program is written in a language that does not support the CDD, you must redefine the record in the program.)

All data used in requests must be defined in the CDD, including data entered by the terminal operator (which TDMS moves from the form to the record) as well as information that is displayed on the terminal screen (which TDMS moves from the record to the form).

The record definer creates record definitions using either the VAX CDD Data Definition Language (CDDL), VAX DATATRIEVE, or VAX DBMS. As with form definitions, record definitions are stored in the Common Data Dictionary.

Form Definitions

The form definition describes the image that is displayed on the terminal at runtime. A TDMS application uses forms to collect information from, and display information to, the terminal operator. The form definition contains the information that identifies:

- The screen image of the form. The screen image includes the location of background text and fields as well as video highlighting. (Background text is text that is always displayed when the form is displayed; fields are locations on the form where data can be collected or displayed.)
- A set of attributes for each field on the form (including means for validating data).
- The location, length, and picture-type of each field.
- The location of scrolled regions on the form.
- The name of a Help Form, which the terminal operator can display at runtime.

Terminal Operator

The name *terminal operator* identifies the individual who uses the TDMS application at runtime. The terminal operator may be entering information (into the application database), reading information generated and displayed by the application, or both.

Requests

Requests are English-like, self-documenting statements. While very easy to follow, they can be quite powerful in terms of the work they accomplish, the coding they replace. Requests can replace virtually all of the terminal input and output coding, that would otherwise be required of the application program, with high-level, nonprocedural definitions. A typical simple request identifies a form to be displayed and the data to be collected and/or displayed on the form. Using requests, TDMS permits data that is to be collected (or displayed) on a single form to be sent from (or to) any number of records. Requests provide for the moving and mapping of data type conversions, plus the automatic handling of scrolled fields. More complex requests provide additional capabilities, such as the inclusion of conditional instructions. The conditional logic in a request allows:

- forms selection from a list of possible forms
- the handling of error conditions
- the dynamic modification of forms presentation characteristics, such as blinking and bolding

The application program is therefore independent of the form data input/output process. Its primary functions are to:

- Call and execute requests
- Provide access to the database that the application uses
- Provide error-processing functions that avoid data corruption and runtime errors

The application programmer does not need to be concerned with mapping data between forms and records, since this is done entirely by the request. In many applications, TDMS can reduce the number of programming statements and error messages from the application program, using requests to undertake these functions.

The key element in a TDMS application, the TDMS request controls the information that is displayed on the terminal and collected from the terminal operator. By defining the terminal input and output, the request replaces major portions of code that otherwise would have to be included in the application program.

As the result of instructions specified by a request, TDMS can:

- Display a form
- Allow data to be entered on the form and transferred to the record
- Allow data to be transferred from the record and displayed on the form
- Conditionally perform the above operations based on a value previously entered by the terminal operator or returned by the program
- Allow the operator to use program request keys that can return predefined data to the record

TDMS requests are created by the VAX TDMS *Request Definition Utility* (RDU), which also stores the requests in the CDD.

Request Library Definitions

A *request library definition* is a list of one or more requests. To be used in a TDMS application, a request must be named in at least one request library definition.

The request library definition is created by the Request Definition Utility, and it is stored in the CDD. You can use more than one request library definition in an application.

Request Library Files

A *request library file (RLB)* is a VMS file based on a request library definition. The RLB includes:

- Any requests named in the request library definition
- Any form definitions identified in each request
- Any record definitions identified in each request

You use the Request Definition Utility (RDU) to build an RLB. If a definer modifies a request, form definition, or record definition after an RLB has been built, you must rebuild the RLB in order to incorporate the changes into the application at runtime.

At runtime, the program can execute a request *only if* the request is in an RLB file and the RLB file has been opened by the program. When the program executes a request, the request, record information, and form definition are retrieved from the RLB file, not from the CDD.

TDMS Utilities

VAX TDMS provides two utilities to create, store, and modify definitions: the TDMS Form Definition Utility (FDU) and the TDMS Request Definition Utility (RDU).

The TDMS Form Definition Utility

The TDMS Form Definition Utility (FDU) provides all of the capabilities that the form definer needs to create or modify form definitions and store them in the CDD. Using the FDU form editor that is included in the FDU, the form definer can:

- Create a screen image of the form, including:
  - Background text (text that is always displayed on the form)
  - Fields (locations on the form where data can be collected or displayed)
  - Video highlights that can be activated whenever the form is displayed and/or when a field is available for operator input
  - The screen background (dark or light) and number of columns (80 or 132)
- Assign specific attributes, validation procedures, and input order to fields

The TDMS Request Definition Utility

The TDMS Request Definition Utility (RDU) provides all of the capabilities that the request definer needs to:

- Define and modify requests and store them in the CDD
- Define and modify request library definitions and store them in the CDD
- Build request library files

RDU includes a validation procedure to ensure that:

- The syntax used in each request is valid
- Each form definition named in the request exists as a CDD location containing that form definition
- Each record definition named in the request exists as a CDD location containing that record definition
- The data mappings between form and record definitions are consistent with TDMS data mapping rules

RDU also creates, validates, and stores request library definitions and verifies that each request named in a request library definition exists in the CDD.

RDU is also used to build request library files (RLBs). To build the RLB file, RDU retrieves the requests named in the request library definition and the form and record information identified in each request. As output, the RDU generates the RLB in a VMS directory. When building the RLB, RDU validates each request to make sure that form and record definitions exist, that the request syntax adheres to RDU syntax rules, and that all input and output mappings are legal.

(The definer can choose to deactivate the RDU validation procedure when creating or modifying requests and request library definitions. RDU always validates request library files and does not build an RLB if it detects serious errors.)

In summary, the request definer uses RDU to create and modify requests and request library definitions, and to build a request library file. TDMS can execute a request only when the request (and the form and record information identified in the request) is included in an RLB file. RDU provides a validation procedure that may be used when creating or modifying requests and request library definitions and is always used when building an RLB.

Summary — VAX TDMS Application Elements and Utilities

Form definitions describe the appearance of the screen at runtime and regulate the type of data that the terminal operator is permitted to enter at runtime. *Record definitions* describe the structure of the data that is used in the application. *Requests* usually identify form definitions, record definitions, and the input and output mappings that take place between them. Form definitions, record definitions, and requests are stored in the Common Data Dictionary, using utilities provided by VAX TDMS and VAX CDD.

Request library definitions list the requests used in an application. A request library file (RLB) is based on the requests named in a request library definition; the RLB includes each request and the form and record definitions named in each request. A request must be in a request library file that has been opened by the program in order to be used in a TDMS application.

At runtime, TDMS programming calls execute requests stored in RLB files. The application is controlled by the requests that the program executes and the instructions specified by such requests. Request instructions can direct TDMS to display forms, collect and/or display information, or perform conditional operations.

▪ Using VAX TDMS with Other Products

Other than the VMS operating system, TDMS's only prerequisite software is the VAX Common Data Dictionary. TDMS can optionally be used in conjunction with VAX DATATRIEVE, VAX DBMS, VAX Rdb/VMS, and VAX ACMS, it being a prerequisite for ACMS.

The VAX Common Data Dictionary (CDD)

The CDD is the facility in which all form definitions, record definitions, requests, and request library definitions must be stored in order to be used in a TDMS application. The CDD is not used to store data.

Before generating the CDD definitions for TDMS applications (record definitions, form definitions, requests, and request library definitions), you should understand the structure of the CDD and know where in the CDD you should store your definitions.

In general, the CDD is organized into a hierarchy of directories, similar to VMS directories and subdirectories. Definitions used by TDMS are stored in *dictionary objects*; the name of the definition indicates the path used by the CDD to locate the dictionary object. For example, a form definition named EMPLOYEE.DISPLAY.FORM is found in the CDD by following the path of (dictionary directory) EMPLOYEE to (dictionary directory) DISPLAY to (dictionary object) FORM.

Form definitions and requests are stored in the CDD using the Form Definition Utility and the Request Definition Utility. To store record definitions in the CDD, use the CDD Data Definition Language (CDDL), which is included as part of the VAX Common Data Dictionary software. Examples of record definitions are shown in the *VAX TDMS Sample Application Manual.* For complete information concerning the use of CDDL, see the *VAX Common Data Dictionary Data Definition Language Reference Manual.*

Language Support

If an application program is written in VAX COBOL or VAX BASIC it can, at compile time, use the "copying" features of those languages to include record definitions directly from the CDD. Applications written in other languages must define records in the application program itself and in the CDD.

Applications can be written in any VAX native mode language that adheres to the VAX Procedure Calling and Condition Handling standard. Specific languages supported are:

VAX BASIC	VAX COBOL
VAX FORTRAN	VAX PL/1
VAX C	VAX DIBOL
VAX BLISS-32	VAX PASCAL

VAX FMS

The VAX FMS Forms Management System is designed to aid in the development of application programs that use video forms. FMS manages these forms for application programs that use Digital's VT100- and VT200-compatible terminals. Forms defined using VAX FMS provide a programmer with the ability to use the following features of those terminals:

- Individual character attributes of reverse video, bold, blinking, and underline
- Line attributes of double-width, double-height, and scrolling
- Screen-wide attributes such as 80- or 132-column lines and reverse video
- Alternate character sets including the VT100 "special graphics character set" for line drawing

▪ Who Uses VAX FMS

As you might expect, users who select FMS as their terminal/forms management solution do so for very different reasons than those cited by TDMS users.

VAX FMS should typically be selected when:

- There are requirements for extensive validations of data at the field level.
- There are requirements for upward system compatibility across operating systems.
- There is no need for the VAX CDD.
- There is a need for field by field interaction between the application program and the form.
- Screen-based applications are to be developed for use in the ALL-IN-1 environment.

▪ The Benefits of VAX FMS

- FMS is easy for your data entry personnel to use. Many of its features and capabilities make it possible for your form screens to be varied, attention-getting, and conducive to minimal fatigue and/or error tendencies.
- FMS is easy for your application programmers. It takes some of the workload off your programmers by taking forms design responsibility out of the application program. With FMS, you store video forms in their own forms libraries. You can modify old forms and create new ones without disturbing application programs. Once the form driver programs are written into your programs, there is no need to recompile or relink whenever a form needs to be changed or created.
- FMS is easy for your forms designers. Virtually all of your computer users can be trained to design their own forms with VAX FMS. The interactive Forms Editor lets you see your forms as you design them on a VT100- or VT200-compatible terminal, making changes as you go.

▪ A Closer Look at VAX FMS

The Interactive Forms Editor

The VAX FMS interactive Forms Editor is the actual tool you use to design your forms on the terminal screen. It provides a simple means of entering, storing and modifying FMS form descriptions. Being interactive, the editor lets you see each form as you create it, exactly as it would appear in finished form. You can make adjustments, or wholesale rearrangements, at any time during the design process.

The Forms Editor is also menu driven, with an extensive online HELP facility, making it simple enough for end users to successfully create and modify their own forms.

The Forms Test Facility lets you test as many possible matrices as you wish before selecting the most preferred final form design. And using your terminal's keypad, a variety of editing functions can be invoked. All form attributes, individual field attributes, and named data constants are assigned in this form editing process. The result of a session using the Forms Editor is an intermediate form file, ready for processing by the Form Librarian.

The Form Librarian

Essentially what its name suggests, the Form Librarian allows individual forms to be inserted (into storage), deleted, extracted and replaced. Forms will typically reside in form libraries on disk, and are retrieved as needed by application programs during their execution. This arrangement makes possible a high degree of independence between form data structures and application programs. Forms can be modified without the often-encountered requirement for recompilation or relinking of those application programs.

User Action Routines (UARs)

A key feature of VAX FMS is the virtually boundless capabilities it offers through its User Action Routines, or UARs. Functioning as extensions to the FMS Forms Driver, UARs are routines that you can write in your favorite VMS programming language and then associate with a field, a HELP request, or a function key. UARs can be as long and sophisticated, or as short and simple, as your needs dictate.

The Forms Driver calls a UAR "into action" when a specified event takes place, a field is completed, or a function or HELP key is pressed. Different forms in different applications can share one or more UARs. Regardless of the VMS language(s) in which they are written, all UARs can be stored in UAR libraries, available to application developers.

The three different types of UARs available are Field-Completion, HELP Key-Associated, and Function Key UARs.

Field-Completion UARs

You can associate a UAR with the completion of a particular field. As soon as data entry to that field is completed, The Forms Driver calls the associated UAR(s) before advancing to the next field, giving you considerable flexibility in data validation.

As an example, suppose you had a form with DATE-OF-BIRTH and AGE fields. You could write a UAR that would compare the DATE-OF-BIRTH with the current date to validate each person's age, and would then associate this UAR with the completion of the AGE field. Every time a user filled in the form, the UAR would automatically validate the AGE field.

You can also use UARs associated with field completion to branch from one form to another. If you had a number of forms that needed to be filled out sequentially, you could associate a UAR with the completion of the last field in the first form to call the next form in the sequence.

HELP Key UARs

You can associate a UAR with a HELP request to provide even more detail than is provided by standard FMS HELP processing. You can also use this feature to track HELP requests.

You can write a UAR that presents a unique HELP message for each field in addition to the standard FMS help. For example, you could display a list of allowable entries for a particular field. Or you could, of course, defer to the form-level help for those fields that do not require more detailed help messages.

You can also use the HELP key-associated UARs to keep track of HELP requests. You could track requests by user, by form, by field, or by any combination of the three. In this way you could quickly discover if users were having particular difficulty with a certain form, or even a specific field.

A UAR that is activated before the standard field- or form-level HELP facility is called a PRE-HELP UAR. You can also write a UAR that will be activated only after the form-level HELP messages have been exhausted — this is called a POST-HELP UAR. This POST-HELP UAR could provide a more detailed HELP message or could even notify a supervisor that the user has a problem.

Function Key UARs

You can associate UARs with function keys to define function keys for your application, which can make that application exceptionally easy to use.

For example, if you were designing a hotel management application you might have separate forms for reservations, check-ins and check-outs. You could establish UARs to call these forms with different function keys. You could then label certain keys accordingly, speeding the access of the correct form to the desk clerk's screen.

The Forms Driver

The Forms Driver is the runtime component of VAX FMS. Called from application programs as the means to control screen processing, the Forms Driver is a shareable, native mode subroutine. It manages all screen I/O, displays forms, manipulates the screen, performs some basic input validation, and responds to an operator's request(s) for HELP. It allows you to specify input and output one field at a time or on a whole-form basis.

FMS's Forms Driver has several features that enable fine tuning of the interaction between application and user. Field Highlighting, for example, causes the video attributes of each field to be changed as the cursor enters the field, and to be restored to their original state when the operator exits that field.

Named Data

VAX FMS permits up to 65,000 items of Named Data per form. Named Data allows information needed by an application program — such as forms linkage, operator messages and other human language information, and data validation criteria — to be defined, stored and modified independently of the application program.

The Forms Language

The Forms Language Translator, optional with FMS, allows you to create forms from any type of terminal, even a hardcopy terminal. Use the text editor of your choice to create a text file that contains the form language description of the form. You can then translate the text file into a binary form file using the Forms Language Translator.

FMS's Forms Language itself is logical and easy to learn. It has only ten statements, yet gives you all of the design flexibility that you have with the FMS Forms Editor.

Application programs can generate form description in the form language. Suppose, for example, you had a large number of forms with your company name at the top of each form. You then wanted to change just the company name in each form. You could write an application program to read in, and then modify, the form language description of each form.

Supported Languages

FMS can be used in conjunction with applications written in VAX COBOL, VAX BASIC, VAX FORTRAN, VAX PASCAL, VAX C, VAX PL/1, VAX BLISS-32, and VAX DATATRIEVE.

TDMS AND FMS – RELATIVE STRENGTHS

As was noted at the beginning of this chapter, VAX TDMS and VAX FMS are distinct yet similar products. They are each a significant part of Digital's commitment to provide VAX video forms products that meet the needs of all customers. Each will continue to evolve further, adding enhancements, while retaining its own particular advantages as a terminal forms solution.

A number of generalizations can now be stated here, summarizing each product's comparative advantages, and they will then be expanded into greater detail in the remainder of the chapter.

TDMS will typically be the product of choice for users of other VAX Information Architecture tools. It will most often be associated with the forms management aspect of applications that are fairly complex to develop, as it can help to reduce that complexity. It provides a fourth-generation, nonprocedural definition language for defining the exchange of data between application program and screen. TDMS is a record level interface. It requires the VAX Common Data Dictionary, and is required if VAX ACMS is to be used.

FMS is recommended where there is a need for extensive validation of data at the field level, as opposed to TDMS's record level orientation. FMS applications, then, often require large numbers of fields per form, and require interaction with those applications on a field by field basis. FMS also tends to be the solution where there are requirements for upward system compatibility across operating systems, where there is no need for the VAX CDD, and where a wide variety of terminal types might be used. Finally, FMS is the forms management tool for ALL-IN-1 users developing screen-based applications.

In the event that a substantial mix of both types of applications is involved for a given customer, both products can be used and can successfully coexist. Note that VAX DATATRIEVE supports forms created with either product, so the TDMS/FMS decision should be made solely on the basis of applications that are to be developed and the environment in which they will be run. Now, recommendations based on those criteria in further detail.

When TDMS is the Best Solution

VAX TDMS is a fourth generation programmer productivity tool that typically uses third generation languages for application processing and database management.

It should be selected when:

- There is a need for programmer productivity to reduce the lifecycle costs of developing and maintaining complex VMS applications.
- There will be other VAX Information Architecture products used, such as VAX CDD, VAX DATATRIEVE, VAX Rdb/VMS or VAX DBMS.
- There will be a need to migrate to VAX ACMS and transaction processing for the implementation and control of applications that are more sophisticated and complex than those normally implemented with TDMS.

With its record level interface, TDMS provides terminal and data independence, freeing applications of the need to write procedural code to perform individual field validation, data mapping and conversion, and the manipulation of scrolled regions. Applications using TDMS may be developed with any third generation programming language that adheres to the VMS calling standard. The applications need only be concerned with the logic to perform application-specific processing and to control the flow between the screen and the database. TDMS is a prerequisite product for ACMS, providing the terminal handling subsystem for ACMS-developed applications. Being an integrated member of the VAX Information Architecture, TDMS makes use of DBMS, RMS and DATATRIEVE record definitions, transparently mapping screen data to database record definitions stored in the CDD. The use of the CDD as a central repository of TDMS definitions significantly reduces the maintenance of applications as these definitions can frequently be modified without affecting the application program. TDMS is best suited for those applications that process screen data a record at a time, as contrasted to the field by field orientation of FMS.

When FMS is the Best Solution

FMS should be selected when:

- There are requirements for extensive validations of data at the field level.
- There are requirements for upward system compatibility across operating systems.
- There is no need for the VAX CDD.
- There is a need for field by field interaction between the application program and the form.
- Screen-based applications are to be developed for use in the ALL-IN-1 environment.

FMS is preferable for those applications that require a large number of fields per form, and require interaction with the application on a field by field basis. This is contrasted with the record level processing of TDMS, where interaction with the application is typically performed after completion of all data entry operations for all input fields on the form. A simple example might be a user form requesting part number, description, quantity, price, back order amount, and total. The field level programming interface of FMS more easily allows the application program to read the part number and verify (from an FMS User Action Routine) that the part number is active and represents a part that is both valid and in stock before prompting the operator for the remaining variables (description, quantity, price, etc.). FMS would also be preferable if the application required the inputting of multiple line items of similar part numbers and variable data and required that the operator have the ability to transparently traverse the screen, (up/down arrow keys, backspace, etc.) modifying and validating fields that had been incorrectly entered. Although this kind of application is possible to handle using TDMS, it is not as efficient as with FMS. FMS may be used with most VAX programming languages as well as with VAX DATATRIEVE.

Chapter 8 • VAX ACMS

VAX ACMS, the VAX Application Control and Management System, was designed to reduce the lifecycle costs involved in designing, developing, maintaining and controlling transaction processing and other complex VAX/VMS applications.

Unlike traditional application development tools, ACMS allows for the replacement of large amounts of application code with high level definitions stored in the VAX Common Data Dictionary. With the use of such definitions, users now have available a fourth generation-like language facility that can significantly reduce the development and maintenance lifecycle costs of large, complex software projects.

VAX ACMS is comprised of two major functional parts. The first is used to design, develop and maintain transaction processing and other complex applications using a fourth generation-like language. The second part is used to monitor and control both applications developed under the first part and those developed with other VMS tools.

Examples of applications appropriate for the use of VAX ACMS include:

- Operations support, such as order administration, personnel administration, inventory control, and scheduling
- Inquiry and information retrieval, such as accessing customer or employee records for quick reference or for providing decision support
- Accounting, including accounts payable and receivable, funds transfer, foreign exchange, and payroll

ACMS is ideally suited for, but by no means limited to, use in "transaction processing" applications. Processing environments such as payroll, inventory, order processing, airline reservations and on-line banking are generally considered the most "typical" transaction processing applications. ACMS, however, can prove to be an excellent solution in any application where business procedures can be broken down into some form of individual events, or units of work.

ACMS is aimed primarily at those applications which will take three or more "programmer-years" to develop, and which are regarded in the industry as being in the transaction processing class of applications. Furthermore, applications for which ACMS is suitable will typically be highly structured, support large numbers of users, be used in a production environment critical to an organization's daily operations, and will be used with large, complex databases. They will also tend to be highly subject to change, and often need to run on non-dedicated systems.

Many studies estimate that 60% to 80% of the cost of application programming is incurred in maintaining applications after they have been developed. One of the goals of ACMS is to reduce those costs.

A key principle behind ACMS is the separation of application development from application use and control. More specifically, application characteristics that are likely to change often, such as which users can run which tasks, should not be the concern of development personnel. VAX ACMS therefore both provides the tools to design, develop and maintain on-line applications, and also separately helps you to create the operational environment in which they run.

With VAX ACMS you can:

- Describe, using a high-level definition language, the work or task(s) a user needs to do.
- Define access and control characteristics for users and devices
- Set up or change menus that let terminal users easily select the task they want to run
- Employ terminal user commands to run tasks, get help information, and exit ACMS
- Control what resources are available to process the tasks in an application
- Monitor and record system activity, application use and performance
- Change system parameters at runtime, "tuning" your usage of system resources
- Control the startup and shutdown of applications

▪ Who Uses VAX ACMS

VAX ACMS addresses several different types of users, all of whom are in some way involved with complex, online applications:

- *Application designers*, responsible for the overall specification and definition of an application and its tasks.
- *Application programmers*, responsible for creating and maintaining the programs for that application.
- *Application managers*, who set up menus, define applications, control user access to applications and tasks, and monitor and maintain applications.
- *System managers*, who authorize users and terminals for access to ACMS, and ACMS applications for access to VMS.
- *Terminal users*, who select and run tasks from ACMS menus.
- *ACMS operators*, who control and monitor the day-to-day operations of ACMS applications.

Note that these terms represent roles, not job titles. In many organizations, a single person performs more than one of these roles.

▪ Benefits of VAX ACMS

- ACMS forces application design work at the outset of a project. *The better the original design, the less rework tends to be needed.*
- ACMS's task-oriented, modular solutions allow the sharing of individual pieces (tasks) among multiple applications. *The "ripple effect" of necessary changes is minimized by this modularity.*
- Debugging need not wait until an entire application is coded. With ACMS, programmers can debug task by task, as the tasks become complete. *Errors can be detected and corrected in early development.*
- ACMS provides a high level definition language, *requiring much less coding than with traditional application development tools.*
- ACMS is an integrated member of the VAX Information Architecture. It uses TDMS to handle terminal I/O and VAX DBMS to provide automatic database journaling and recovery.
- *These advantages, individually and taken in total, can yield a significant increase in programmer productivity.*
- The system manager can exercise some control over users' terminals.
- Designated terminals can bypass VAX/VMS login procedures.
- Each image is started only once, then shared by all its users.
- Users of an ACMS application share a number of central ACMS processes, rather than each needing their own. Although the overhead of these processes is higher than that of VMS processes, this cost is outweighed by savings in system resources as the number of users increases.
- *These features enable an improvement in the efficient usage of VAX system resources.*
- Both development and system management can control access to applications at different levels.
- The system manager can optimize performance of an ACMS application by:
  - Monitoring use by user or task
  - Increasing or decreasing the number of processes
  - Setting priorities, changing quotas, limiting functions.
- *So, both the developer(s) and the system manager can control the application(s) via standard utilities.*

▪ A Closer Look at VAX ACMS

Application Development

VAX ACMS provides tools for developing complex on-line applications. Applications developed using ACMS then also run under its control.

One critical goal of ACMS is to reduce application maintenance costs and increase programmer productivity without sacrificing efficient use of system resources. It accomplishes this by providing a way of implementing the tasks in an application that is different from those provided by VMS or by the other VMS-layered products.

In contrast to traditional application development tools, which require substantial programming knowledge, ACMS lets you replace large portions of program code with high level definitions. These definitions, called *multiple-step tasks*, are simple, direct statements. ACMS provides Application Definition Utility (ADU) clauses for creating task definitions which, like other ADUdefinitions, are stored in the CDD.

There are two types of steps in a multiple-step task. An *exchange step* handles terminal I/O, usually by means of a VAX TDMS request. The request uses forms for input and output. A *processing step* does the computation or database work needed by the task. It uses a subroutine or procedure written in a programming language such as VAX COBOL or VAX BASIC, a VAX DATATRIEVE command or procedure, a DCL command or procedure, or a VAX/VMS image. At the end of each step you can define one or more *actions* that determine what the task will do next. Interaction between the exchange step and the processing step is handled via workspaces that are passed between steps.

Figure 8-1 shows the definition for a simple task that writes a new employee record to a file. The task first calls a TDMS request that asks the user for information about the employee. When the information has been entered, the task calls a program to write that information to a file. If an error occurs in writing the information, the task returns to the exchange step to display an error message.

```
CREATE TASK ADD_EMPLOYEE
  WORKSPACE IS ADD_EMPLOYEE_WORKSPACE;
  BLOCK WORK
    EXCHANGE
      REQUEST IS GET_EMPLOYEE_INFORMATION
        USING ADD_EMPLOYEE_WORKSPACE;
    PROCESSING
      CALL PERSADD IN PERSONNEL_SERVER
        USING ADD_EMPLOYEE_WORKSPACE;
      ACTION
        CONTROL FIELD IS PERSADD_RETURN_STATUS
          "ERROR"   :  GO TO PREVIOUS EXCHANGE;
          "SUCCESS" :  EXIT TASK;
        END CONTROL FIELD;
  END BLOCK WORK;
END DEFINITION;
```

Figure 8-1 ▪ Multiple-Step Task Definition

Multiple-step tasks use *workspaces* to pass information between steps. In the definition shown in Figure 8-1, the workspace named ADD_EMPLOYEE_WORKSPACE passes information from the exchange step to the processing step.

Tasks developed using ACMS can use either VAX databases, (VAX DBMS or VAX Rdb/VMS), or RMS files. If a task uses VAX DBMS recovery actions can be controlled by the task definition, further simplifying the development and maintenance of the application. ACMS uses VAX DBMS facilities to provide automatic journaling and recovery.

Structuring the task into terminal I/O steps and processing steps makes the task definition easier to understand and maintain. In addition, the separation of terminal I/O from processing lets ACMS dedicate different, specialized VAX/VMS processes to each kind of work. ACMS system processes can be used to handle the terminal I/O for many users. Another kind of process, called a *server*, is defined to handle the processing steps. A server can be dedicated to computation, database interaction, or other processing work.

Server processes can be used by many processing steps without having to be started and stopped for each task. A server process can handle the processing step for one task while other tasks do terminal I/O; the same server process can handle processing for a second task while the first task does terminal I/O.

There are two kinds of servers:

- *DCL servers* handle images, DCL commands, DATATRIEVE commands, DATATRIEVE procedures, and other processing that can be run from DCL command mode.
- *Procedure servers* handle only subroutines written in VAX COBOL, VAX BASIC or other VAX languages.

Procedure servers are more efficient than DCL servers. For example, procedure servers let the application perform work common to many tasks, such as opening files, just once when the server process is started.

Because servers can be used by many tasks, they are defined in a *task group* definition. The task group defines resources that can be shared by many tasks. These resources include TDMS request libraries, VMS message files, ACMS workspaces, and servers.

In addition to the ADU clauses for defining tasks and task groups, ACMS provides two facilities to help the application programmer develop ACMS applications. The *Task Debugger (ACMSDBG)* lets the programmer debug tasks without setting up applications and menus. With the Task Debugger, an application programmer can start servers and tasks. While a task is running, the programmer can set breakpoints, examine and change workspace contents, and use the VAX Symbolic Debugger to control processing steps. The commands and qualifiers are similar to those of the VAX Symbolic Debugger.

Runtime Control and Management

ACMS also provides tools for defining, monitoring, and controlling on-line applications. Although designed to supply the operational environment for tasks defined in ACMS's application development phase, the runtime control and management portion of the product can also be used to monitor and control existing applications running under VAX/VMS. So while use of the "full ACMS" product is recommended in the majority of situations, the runtime control and management portion alone is frequently used to create a runtime system on a target node to execute applications developed on a central node.

Using VAX ACMS involves the following:

- Using menus for easy access
- Controlling application availability
- Controlling access to applications
- Monitoring application use and performance

- USING MENUS FOR EASY ACCESS

  The central ACMS control and management facility is the Application Definition Utility (ADU). This utility provides a set of English-like clauses for defining menus and for defining the operational characteristics of ACMS applications. For example, one of the characteristics definable with ADU is which users can select which tasks in an application. These *access control lists* can be the same for all tasks in an application or can differ from task to task.

  Application definitions are stored in the VAX Common Data Dictionary (CDD) so they can be easily maintained and controlled.

- CONTROLLING APPLICATION AVAILABILITY

  ACMS includes an Operator Facility, a set of commands for controlling applications. For example, with these commands an ACMS operator can start an application, making its tasks available to users. Or the operator can stop the application so that the tasks are not available and the application does not tie up any system resources. Other ACMS Operator commands display information about applications, tasks, users, and ACMS components. The commands are similar to the VAX/VMS Digital Command Language (DCL) commands an operator would already know.

- CONTROLLING ACCESS TO APPLICATIONS

  The User Definition Utility defines which authorized VAX/VMS users can log in to ACMS. It also defines which menu the user sees upon logging in to ACMS or, alternatively, defines the user as an experienced one who sees a selection prompt rather than a menu after logging in.

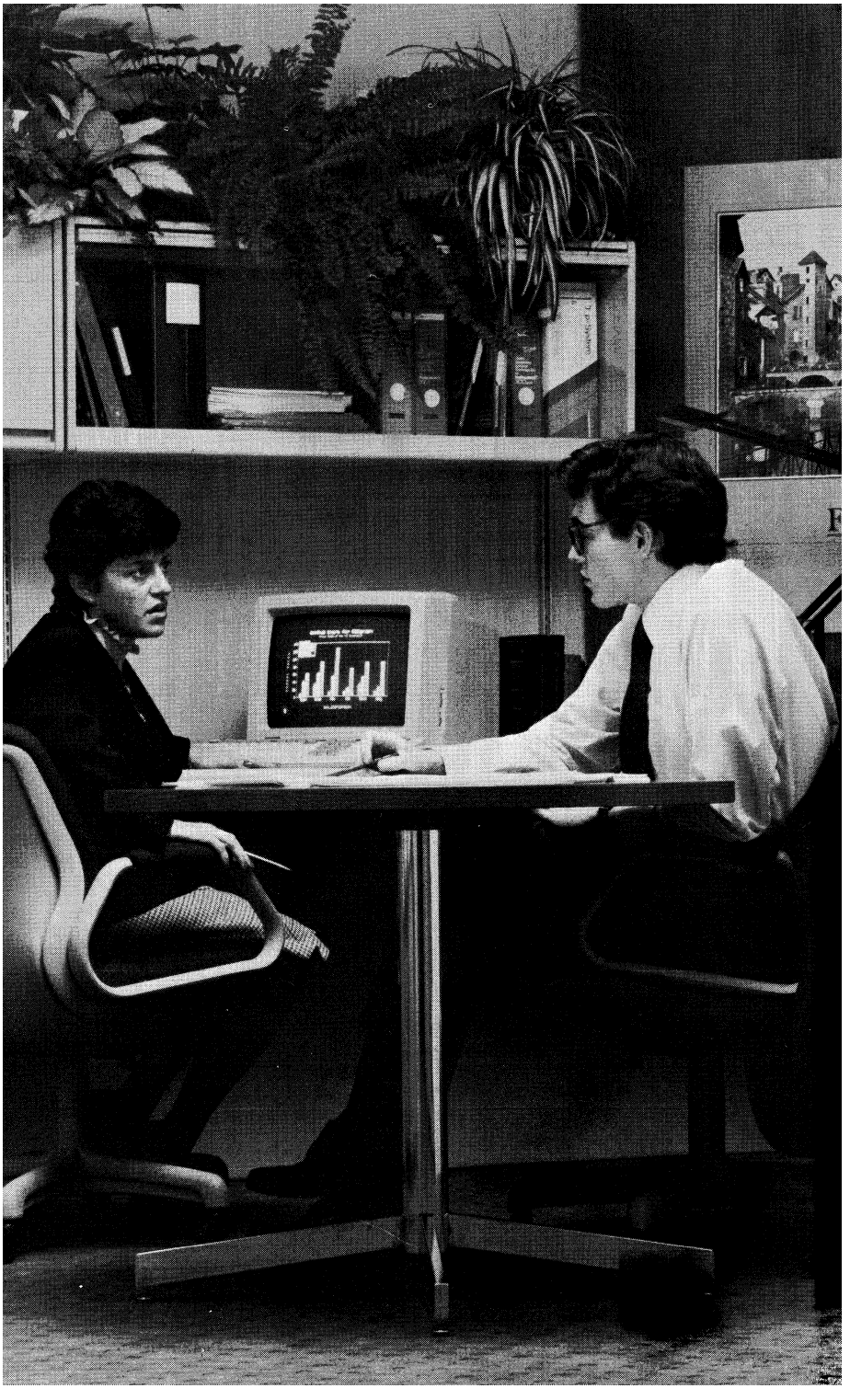
  The Device Definition Utility specifies which terminals can access ACMS and whether or not those terminals log directly in to ACMS. With these utilities, users and terminals can be restricted to ACMS, restricted to VAX/VMS, or given access to both ACMS and VAX/VMS. The utilities are similar to the AUTHORIZE Utility, the system management tool provided by VAX/VMS for authorizing VMS users.

  Although ACMS provides a standard menu format, the format can be modified to suit the needs of different terminal users. In addition, terminal users can select tasks by typing entry names after the "Selection:" prompt, without displaying menus. Certain *terminal user commands* provided as part of the terminal user interface let users display or bypass menus. Other terminal user commands let users get help on using ACMS menus, cancel active tasks, and exit from ACMS.

- MONITORING APPLICATION USE AND PERFORMANCE
  This ACMS component helps ACMS operators and application managers monitor the use of ACMS. An Audit Trail logging facility gathers information about task selections, user logins, and other events. It records this information in a log file. You can then use the Audit Trail Report Utility to take information from the log file and format it into a report. The report can include all information from the file; it can also be selective, including information about only one user, for example. Similarly, the information gathered by the Audit Trail logging facility can include all applications or can be restricted to one or more applications.

  Another component integral to ACMS, ACMSGEN lets system managers change ACMS system parameters, such as how many users can log in, the user names under which ACMS processes run, and the priorities of those processes.

- ACMS RUNTIME SYSTEM
  The final major component of VAX ACMS is the runtime system itself, which uses specialized VAX/VMS processes to handle the menus and tasks in ACMS applications.



Chapter 9 ▪ VAX DECgraph

VAX DECgraph is the VAX Information Architecture's interactive, menu-driven tool for generating graphs from data. It is designed to be used by experienced computer users and novices alike, offering a wide spectrum of capabilities for producing professional quality graphs.

With VAX DECgraph, you can:

- Begin creating your own graphs successfully in approximately one hour.
- See the results of your graph designing immediately — make changes interactively at any point in the design process.
- Obtain data for your graphs in any one of three ways: from a database, from a formatted ASCII file, or through keyboard data entry.
- Produce six different basic types of graphs.
- Enhance your graphs with additional "fine-tuning" features.
- Print, photograph, or send your output to other graphics terminal users.
- Use your graphs for reports, presentations, or "on the spot" decision support.

▪ Who Uses VAX DECgraph

VAX DECgraph is for anyone with a need to see or show data pictorially. With DECgraph, the capability now exists for such people to generate graphs on their own, quickly and easily. Even the most novice of computer users can now become an accomplished designer of graphs in as little as one hour.

DECgraph, then, should be thought of as a productivity tool for anyone who either makes decisions based on data or is responsible for presenting that data to those decision makers. It allows the professional who "owns" a certain set of data, and is therefore "closest" to it in understanding, to create a graph of that data for maximum effect.

As a software product layered on the VAX/VMS operating system, DECgraph runs on any valid VMS configuration. It can be an effective solution not only in the business world, but in scientific, educational, manufacturing, and governmental environments as well.

▪ Benefits of VAX DECgraph

- DECgraph includes, as part of its user documentation, a self-paced introductory tutorial for new users. It also offers several levels of HELP messages to assist users of various DECgraph functions. And small symbols called "icons" guide users through available options. — *You become a productive user of* DECgraph *very quickly.*
- Icons are used extensively as the means of your selecting DECgraph functional options. — *Icons are easy to use, require very few keystrokes, and make it unnecessary for you to learn any type of computer command language.*
- Six different basic types of graphs are available to you. DECgraph offers graphs in the form of a line, scatter, cluster bar, stacked bar, histogram, and pie. — *You select the graph type that is the most effective for making your point, the clearest for explaining a particular situation.*
- DECgraph automatically designs graphs with the options most appropriate for the graph type you select. DECgraph also lets you modify the design of your graph. For example, you can select different color combinations or change the colors to black and white patterns. — *You spend very little time designing a graph, particularly whenever* DECgraph*'s default assignments are accepted.*
- There are three different ways to input or access the data for your graphs. — *You have the choice of either building data or using data already stored on your system.*

▪ VAX DECgraph — a Closer Look

Using VAX DECgraph involves the following:

- Using icons and other general purpose features
- Accessing or entering data
- Designing and modifying graphs
- Generating DECgraph output

DECgraph's General Purpose Features

- ICONS

  DECgraph is primarily a menu-driven tool. The menu choices are represented as pictures, or icons. Whenever icons are displayed on the left portion of the terminal screen, you can select any function they symbolize by simply highlighting the correct icon and pressing the return key. The four arrow keys (up, down, left, right) are used to move the highlighting to the appropriate icon. Icons eliminate the need for users to learn any type of command language — they let non-technical people enjoy the benefits of using a computer.

- HELP LEVELS

  You can always get help while working with DECgraph. You can select the Help icon on the main menu for an overview of DECgraph's concepts and procedures. You can use the Help key (PF2) first to display the icons' labels and then to get a help message for any menu option or prompt. You can then use this key a third time for a more detailed explanation.

- FILES

  Three different types of files are used in DECgraph to specify your graph. The data set consists of two files that contain the identification information and data for your graph, while a Graph Description file stores its design specifications.

Data Input and Access

- KEYBOARD DATA ENTRY

  Selecting the Keyboard Data Entry method of input allows you to build and store tables of data that did not previously exist in any such arranged format. A logically designed, form-oriented data entry screen shows you exactly what information will be needed in order for you to successfully create graphs. And once data is entered, you can change, delete, or save it for later use.

  Keyboard Data Entry will normally be the chosen input method when you wish to graph data that is not stored on your system in any way. It is typically for special situations, requiring one or more fields of data that are not found in your system's database(s).

- USING VAX DATATRIEVE

  DECgraph can use the capabilities of DATATRIEVE as a means of collecting the data to be plotted. That data can be extracted from RMS files, from VMS databases, or from VAX Rdb/VMS databases, in what is essentially a two-step process.

The first step involves entering DATATRIEVE, from DECgraph's data entry menu, and specifying the current collection of data from which you want to create graphs. As an example, you might want to access a certain subset of data on your system that describes various parameters of a collection of yachts. You enter DATATRIEVE from DECgraph's "DTR" icon, then use a "Find" statement to specify the subset of the Yachts data you wish to work with. (Such as, "find" all yachts having an overall length of more than 40 feet, and call them the Current Collection.)

The second step involves simply entering valid DATATRIEVE field or query names on the Keyboard Data Entry screen, when and where needed. These entries will automatically cause the appropriate DATATRIEVE data to load into the Values fields of the DECgraph data entry screen. Then, as has already been explained, the actual generation of graphs from this point forward is extremely simple.

- THE LOAD FILE OPTION

In this third method of bringing data to the graphing phase, an application program can be written to generate a new, formatted ASCII file. Called a "Load File," this file can then be used by DECgraph to create any of six basic types of graphs as with the previously described methods.

Designing Graphs

- THE SIX BASIC TYPES OF GRAPHS

LINE GRAPH - best suited for depicting comparisons of variables over time, line graphs are a popular way to illustrate trends. Nine different line types are available — up to six lines, each representing one "Y variable," can appear on a single line graph.

CLUSTER BAR GRAPH - cluster bar graphs are normally ideal when two or more variables are to be compared by person, by place, or by item. DECgraph permits the use of up to six such "Y variables." As an example, a cluster bar graph could compare the U.S., European, and Asian sales figures for five different salespeople in a company. Three different bars would appear above each salesperson's name — one each for their U.S., European, and Asian selling records. Those three geographic regions represent three Y variables to DECgraph, and as many as three more could be shown for each salesperson.

STACKED BAR GRAPH - as its name suggests, a stacked bar graph displays up to six variables in a single vertical bar, with horizontal lines (and changes in pattern or color) distinguishing the various sections of the bar from each other. Where a cluster tends to be more visually effective in showing differences *by one or more Y variables,* the stacked bar graph is typically best for comparing the *total numeric values* of bars, i.e. the total height of columns. In the previous example, a sales manager interested only in salespeople's total selling success might prefer a stacked bar graph. The height of each person's bar will be the key concern. The manager who wanted to spot salespeople having problems in their non-U.S. sales efforts would choose a cluster bar graph instead, paying close attention to each person's European and Asian bars.

PIE GRAPH - when your most important graphing need is to show the relative sizes of parts of a whole, the pie graph is an excellent choice. If an organization has six revenue categories, for example, which together comprise all of that organization's revenue, a pie graph can clearly illustrate which sources are bringing in more funds than which others.

SCATTER GRAPH - scatter graphs are often an appropriate way to show a comparison of two or three variables, for example, by different people, geographic regions, or corporate divisions. DECgraph can plot up to six different types of markers, from a selection of ten, on one scatter graph.

HISTOGRAM - histograms are used to illustrate a frequency distribution — one might, for example, show the number of a company's employees that fall within each of ten annual earnings ranges. A histogram resembles a bar graph with no blank space between the bars. The bars are at various heights, with the continuous top line resembling a side view of a staircase. All space beneath this top line is filled in with pattern or color.

- SPECIAL DESIGNING OPTIONS

ASSIGNMENT - the assignment option in DECgraph provides a wide variety of fine-tuning alternatives that can improve the appearance and readability of your graphs. It is important to note that a user does not necessarily ever need to use the Assignment option, or any of its sub-options. DECgraph handles the automatic assignment of colors, markers, and line types that are likely to be pleasing to the eye. It is only for matters of personal preference that users might elect to use the Assignment options.

Selection of the Assignment icon leads you to the following five suboptions.

- *Palette Options* — The user can select from twelve palettes, each of which holds a combination of ten different colors.

- *Color Options* — DECgraph offers you twelve palettes, of ten individual colors each, which have been preselected for optimal design effect. You can not only select the best palette for your needs, but can also change the individual color selections that automatically appear within the chosen palette.

- *Line Type Options* — Nine different line types can be used in DECgraph — solid lines, dots, dashes, and combinations of dots and dashes. Not only does this allow for sufficient variety in the event that there are several lines on one graph, it also gives you the ability to do such things as show actual trends as solid lines, or projections for the future as "broken" lines.

- *Marker Options* — Ten different markers are available for use both in scatter graphs and as points along the lines in a line graph.

- *Pattern Options* — There is a set of ten black and white patterns (cross-hatching, dots, diagonal lines, etc.) that you can assign to your graph instead of colors. The graph could be in any color palette, then changed to show black and white patterns. The pattern is the same for color 1 in palettes 1-10, for color 2 in palettes 1-10, etc.)

BORDER - By default, a graph is designed with a four-sided rectangular border. Using the border option icon, however, you can modify a graph to have only a corner border (left and bottom), or no border at all.

FILL - in line graphs, it can be visually effective to fill in the area beneath the graph's line(s). With the fill option, color can be added beneath any lines in the graph.

GRID - the grid option allows users to insert, as background, dashed horizontal grid lines, vertical, or both. (The DECgraph default is no grid lines.)

ISOLATE - it is often helpful in a pie graph, when one "piece of the pie" is to be noticed as having special importance, to separate it slightly from the rest of the pie. In a pie graph showing sources of a firm's total revenue, for example, the section representing net income might be isolated to help viewers notice it and its size relative to the rest of the pie. DECgraph gives you the option of isolating one or more sections of any pie graph.

LEGEND - some graphs require a legend explaining the meaning of different bars, different patterns and colors, or different line and marker types. But other graphs either do not need a legend or perhaps even are more effective without one. Any graph built with DECgraph can include or omit the legend, at your option.

SCALES - users can either accept the numeric scales and ranges that DECgraph automatically assigns to the horizontal and vertical axes, or can "override" those scales with their own specifications. Not only can you choose between linear and logarithmic scaling, but you have the chance to use different minimum and maximum values from those automatically assigned.

SHADOW - to achieve a three-dimensional effect, you can add shadowing to bar graphs (cluster or stacked). The shadowing can then be assigned different colors.

TREND - in scatter graphs, on occasion you might wish to see a trend line reflecting a regression analysis of a set of points. DECgraph can automatically insert such a line, handling all necessary calculations and "drawing" the line for you. This feature can be particularly useful in projecting where future points might appear on a graph over time.

Output

Finished graphs created using DECgraph can be printed on a graphics printer, photographed, "exported" via the VAX/VMS mail utility, and/or stored on your system's mass storage media by their Graph Description and Data Set file names.

- PRINTING

  Graphs can be printed, in either of two sizes, on any of four Digital black and white printers. Single-size graphs measure approximately 3¼" by 5¾", double-size approximately 6½" by 11½". Model numbers of the supported Digital printers are as follows:

  - Single-size graphs — LA12; LA50; LA34-VA; LA100
  - Double-size graphs — LA34-VA; LA100

Color printers are available through third-party vendors — they must be compatible with the RGB (Red/Green/Blue) color connector on Digital's graphics video terminals.

- PHOTOGRAPHING
  Finished graphs can be made into overhead transparencies, 35mm slides, or prints, in color or black and white. Third-party vendors offer a variety of cameras that can photograph the image shown on the terminal screen. Some produce instant prints — with the others the processing choice is yours as to whether it becomes an overhead transparency, a 35mm slide, or a "conventional" print.

- EXPORTING A GRAPH
  Using the GOLD key with the PF3 key, you can create a graphics output file. this file contains the graphics commands for your graph, letting you display it on your terminal without actually entering DECgraph. You can mail this file electronically. The user(s) receiving the graphs can file it and then display it on his/her graphics terminal or graphics printer.

  You can also use graphics output files with VAX DECslide's slide organizer function. The slide organizer can store your graphs in one or more "slide trays," enabling them to be printed sequentially in a batch job or arranged for presentations to be given.

▪ How Does it Interact With Other Products?

VAX DATATRIEVE

DECgraph can use the capabilities of DATATRIEVE as a means of collecting the data to be plotted. That data can be extracted from VAX RMS files, VAX DBMS databases, or VAX Rdb/VMS databases, in what is essentially a two-step process.

The first step involves entering DATATRIEVE, from DECgraph's data entry menu, and specifying the current collection of data from which you want to create graphs. As an example, you might want to access a certain subset of data on your system that describes various parameters of a collection of yachts. You enter DATATRIEVE from DECgraph's DTR icon, make your "yachts" domain available, then use a "Find" statement to specify that subset of the Yachts data you wish to work with. (Such as, "find" all yachts having an overall length of more that 40 feet, and call them the Current Collection.)

The second step involves simply entering valid DATATRIEVE field or query names on the keyboard Data Entry screen, when and where needed. These entries will automatically cause the appropriate DATATRIEVE data to load into the Values fields of the DECgraph data entry screen.

VAX CDD

If you choose to use DATATRIEVE with DECgraph, it requires the services of the VAX Common Data Dictionary, or CDD. The CDD stores DATATRIEVE domains and procedures.

VAX Data Management Systems

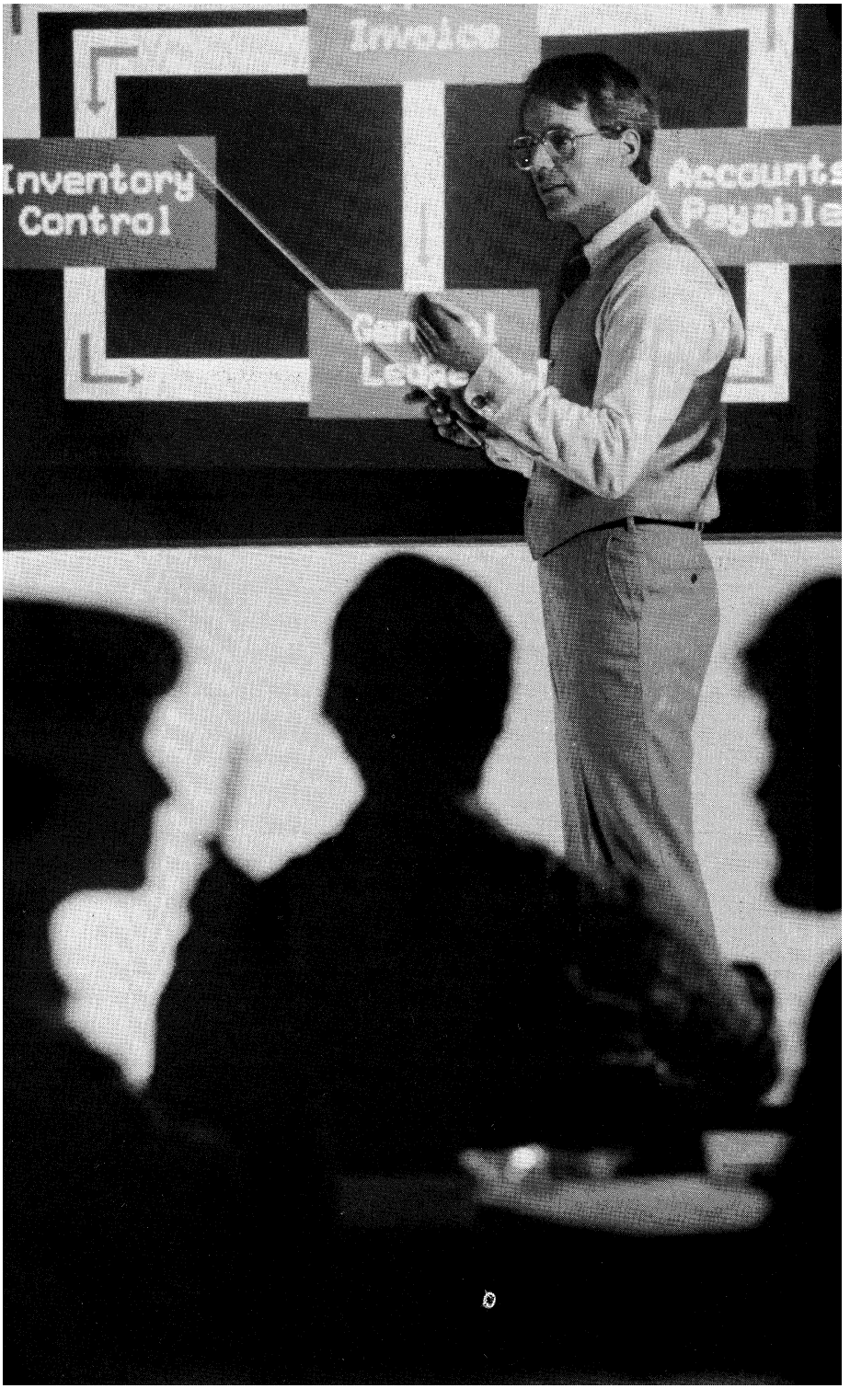
VAX RMS files, or VAX DBMS or Rdb/VMS databases, are all capable of being accessed to provide data for DECgraph. Access is either via DATATRIEVE, explained above, or from an application program written to generate a new formatted ASCII data file called a Load File that DECgraph can use.

VAX/VMS Languages

Any language comforming to the VAX/VMS Language Calling Standard can be used in an application program to provide DECgraph with ASCII data files that it can use.

ALL-IN-1

VAX DECgraph can be run, with all of its capabilities, when called as a menu selection by the ALL-IN-1 Office Menu.


Invoice
Inventory
Control
Accounts
Payable

Chapter 10 ▪ VAX DECslide

VAX DECslide is a menu-driven graphic presentation design tool that runs on the VAX/VMS operating system. It is intended for use by anyone who needs to prepare professional-quality presentations and reports.

With VAX DECslide, experienced and inexperienced computer users alike can:

- Create slides of objects, text, or a combination of both.
- Create the following objects: circle, ellipse, line, square, rectangle, triangle, polygon, and arc.
- Copy, rotate, and move objects.
- Redimension objects or text in height, width, or both.
- Join single objects (shapes) together to form more complex, compound objects.
- Fill objects with patterns.
- Italicize text.
- Add color using one of ten different palettes.
- Print slides on paper, send them to other terminal users, photograph them for use as 35mm slides, or create overhead transparencies.
- Use finished slides for reports, presentations, or "on the spot" decision support.
- Assemble slides into trays for slide shows or printed presentations.

▪ Who Uses VAX DECslide

Anyone with responsibility for creating text and graphic materials for presentations or reports can benefit from using VAX DECslide.

As a layered software product on the VAX/VMS operating system, DECslide runs on any valid VAX/VMS configuration. It can be an effective solution not only in the business world, but also in scientific, educational, manufacturing, and governmental environments as well.

▪ Benefits of VAX DECslide

- DECslide includes, as part of its user documentation, a self-paced introductory tutorial for new users. It also offers multiple levels of HELP messages to assist users in the middle of various DECslide functions. Small symbols called icons guide users through available options. *You become a productive user of* DECslide *very quickly. And the use of icons to select your options means very few keystrokes to get the job done.*
- Because DECslide was designed for use by professionals who might have little or no computer experience, a great deal of art and design work can be handled by those closest to its significance. *There is no longer the need to contract work out to people who might be too far removed from a project to understand fine points of distinction which will make slides successful.*
- Whether DECslide's output is in the form of printed reports, overhead transparencies, or 35mm slides, it has a professional appearance to it. The quality is obvious — it can "do justice" to important material, can convey concepts or information more effectively than can a less professional form of output. To equal this quality without the benefit of VAX DECslide is often very costly in terms of dollars, time, or both. DECslide, *then, can quickly pay for itself in saved design fees which formerly had to be paid out of a department or organization.*

▪ VAX DECslide — a Closer Look

Using VAX DECslide involves the following:

- Using the icons and word list menus
- Creating objects and text
- Modifying the objects and text
- Coloring objects and text
- Outputting slides in one of four ways
- Managing a "library" of saved slides

Using the Icons and Word List Selection Menus

When beginning a DECslide session, a main menu of seven icons appears, making available the following functions:

- STOP — Lets you end a DECslide session.
- HELP — Shows you an overview of DECslide's basic concepts and procedures.
- MESSAGE — Lets you view a list of broadcast messages that have been sent to you during your DECslide session.
- FILE — Accesses a submenu where seven file management options are made available:
  - Exit – Lets you leave the FILE submenu after you have finished selecting your filing options
  - Save – Saves your current slide
  - Delete – Deletes a saved slide
  - Restore – Returns a saved slide to your work area. If a slide was displayed in the work area before you entered the FILE submenu, you can overlay the restored slide onto this slide.
  - Copy – Copies a saved slide into your default directory
  - Export – Creates an export file — a file containing graphic commands which define a slide — for storing via the slide organizer function or for sending to other users.
  - Directory – Displays a slide directory that you specify.
- DESIGN — Clears the screen, displaying your work area — and the current slide, if there is one — letting you use DECslide's keypad keys to create and modify slides.
- PRINT — Accesses a print submenu where you are given three printing options.
  - Exit – Lets you leave the PRINT submenu.
  - Single – Prints your current slide, or any saved slide, in the single-size format.
  - Double – Doubles the printed size of any single-sized slide.
- ORGANIZE — Lets you create, modify, show, and print slide trays.

You select the function that an icon represents simply by using the arrow keys (up, down, left and right) until the desired icon is highlighted. Pressing the Return key then takes you into the chosen function.

Creating Objects and Text

When you are working in DECslide's DESIGN mode, your menus are in the form of word lists across the bottom of your terminal screen. With the arrow keys, you simply move the selection rectangle until it surrounds the word representing the function you want, then depress the Return key. DESIGN mode also features the use of the specialized keypad found at the right of Digital's video terminal keyboards. Each of the eighteen keys on that pad initiates a special function within DECslide. Briefly, those keys enable the following functions:

- The four arrow keys – All allow you to move the cursor, the highlighting rectangle in icons and word list menus, or an object in the edit plane. These keys can also select an object, change its size, rotate it, or draw a line.

  You can use the Gold key in combination with the arrow keys for quick cursor movement in text mode.

- Gold – Pressed in combination with any other key on this keypad, the Gold key enables that key's "second function," — that is, its function shown in the lower half of the keypad legend.

- Help – Shows operating advice for the keypad keys, the iconic menus, the word list menus, and the prompts. "Gold Help" shows a brief description of each of the keypad keys.

- Display – Displays the current slide for viewing or photographing.

- Export – Creates an export file using the current slide.

- Print – Prints the current slide in standard size.

- 2X Print – Prints the current slide in double size.

- Grid On – Displays a background grid in the work area, facilitating the accurate placement of text or objects.

- Grid Off – Removes this grid from the screen.

- Object – Gives you the Object creation word list menu.

- Pixel Toggle – Lets you specify the amount of pixel movement for the cursor, from 50 to 10 to 1 (Coarse, Medium, and Fine respectively). The Pixel Toggle key can also vary the amount of rotation or size change.

- Delete Line/Object – Deletes the line of text the cursor is on, or the object that is in the edit plane.

- Undelete Line/Object – Restores the line or object just deleted.

- Insert/Replace – Lets you switch your text entry from Insert mode to Replace mode. You may use this as a toggle key, switching back and forth between the two modes.

- Paint – Gives you the Paint word list menu.

- Join – Lets you join objects together to form a compound object.

- Separate – Lets you separate subobjects from a compound object.
- Delete C – Deletes the character where the cursor is.
- Undelete C – Restores the character just deleted.
- Change – Gives you the Figure Change word list menu when you are designing a graphic object or the Text Change word list menu when you are working with text.
- Copy – Lets you copy an object (graphic or text) that is in the edit plane.
- Erase – Lets you erase your design work, clearing the screen.
- Accept – Signals to DECslide that you have completed a procedure, such as selecting a menu option, naming a slide, accepting a design operation, or placing new slides in a slide tray.
- Cancel – Cancels the most recent operation you performed.
- Main Menu – Gives you DECslide's main menu.
- Select – Lets you pick up an object from the finished plane to do more design work on it in the edit plane. You may also use this key to indicate the beginning point of a line you are drawing, to join objects into a compound object, or to select objects to be painted. With the slide organizer, you use this key to select slides for trays.

  In Text mode, you use this key to indicate the start of the desired text range for the Zoom, Italic, and Dimension options.

Choosing the "Object" function from DECslide's keypad lets you create any (or all) of the following objects:

- Text – Beginning at the point where the cursor is positioned
- Box – Places a square around the current position of the cursor. (Can be "dimensioned" into a rectangle – see below).
- Circle – Places a circle around the current position of the cursor. (Can be dimensioned into an ellipse).
- Line – Lets you draw a vertical or horizontal line, which you can "stretch" to make diagonal. You can also use this option to draw multisided shapes.
- Polygon – For drawing multisided shapes, this option draws the last side of the shape for you.
- Triangle – Places a triangle around the current position of the cursor.
- Arc – Places a one-quarter-circle arc to the upper left of the current position of the cursor.

Modifying Objects and Text

While in Design mode you use the Change key to modify objects or text. Objects and text have different sets of change options.

There are five basic ways to modify text:

- Edit – Lets you change text after it has undergone any other type of change except rotation.
- Zoom – Lets you increase or decrease the overall size of a selected range of text.
- Dimension – Lets you increase or decrease the height or width of a selected range of text.
- Rotate – Lets you rotate a text object, in either direction, in 45 or 90 degree increments.
- Italic – Lets you italicize a selected range of text (or return italicized text to normal).

Objects can be changed in the following five ways:

- Zoom – Lets you increase or decrease the overall size of a graphic object.
- Dimension – Lets you increase the height or width of an object.
- Rotate – Lets you rotate an object in 15, 45, and 90 degree increments, in either direction.
- Outline – Lets you select one of five line types for use in outlining an object.
- Fill – Lets you select one of ten patterns with which to fill an object. This Fill option is the black and white equivalent of the Paint option, described immediately below and used when a color terminal (VT241) or color monitor (third-party) is available.

Painting Slides

DECslide's keypad Paint key gives you a word list menu with the following options:

- Palette – Lets you view and choose from a selection of ten different color palettes, each of which has ten individual colors.
- Color – Lets you change the default colors in your slide by selecting other colors in your chosen palette.
- Background – Lets you change the default background color of your slide.
- Object(s) – Lets you select the object(s) to be painted in the default color, or the color you select from your chosen palette.

- Status – Gives you a color status report of your slide and any objects in the slide that you specify. This option is helpful if you are not using a VT241 or color monitor while you paint your slide.

Creating DECslide Output

Slides created using VAX DECslide can be printed on paper, photographed, "exported" to other users' terminals VIA the VAX/VMS mail utility, or exported to your own mass storage for cataloguing with the DECslide Slide Organizer utility.

- PRINTING

DECslides can be printed on any of four Digital black and white printers. Single-size slides measure approximately 3¼" by 5¾", double-size approximately 6½" by 11½". Model numbers of the supported Digital printers are as follows:

- Single-size slides — LA12; LA50; LA34-VA; LA100
- Double-size slides — LA34-VA; LA100

Color printers are available through third-party vendors — they must be compatible with the RGB (Red/Green/Blue) color connector on Digital's graphics video terminals.

- PHOTOGRAPHING

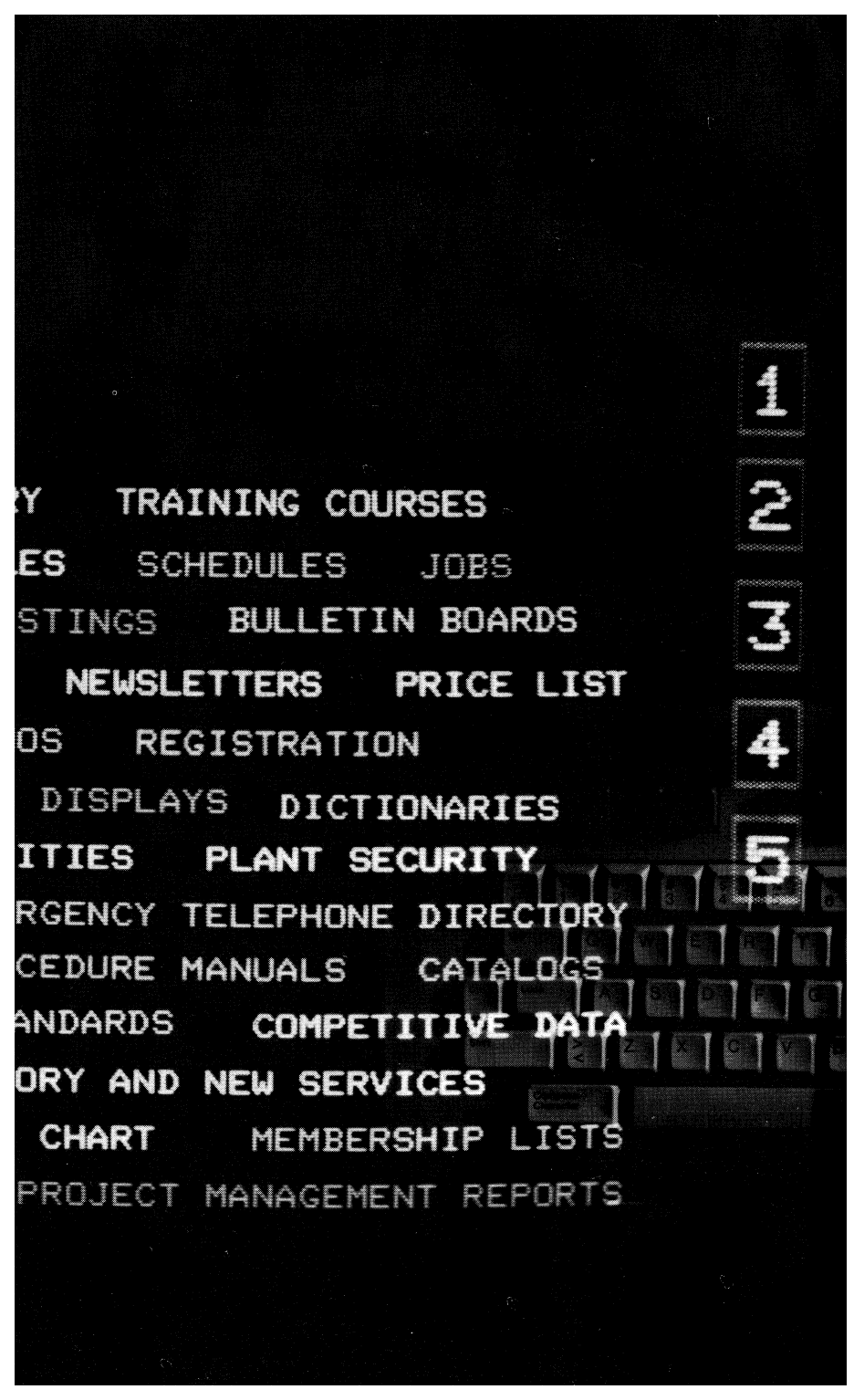
Finished slides can be made into overhead transparencies, 35mm slides, or prints, in color or black and white. Third-party vendors offer a variety of cameras that can photograph the image shown on the terminal screen. Some produce instant prints — with the others the processing choice is yours as to whether it becomes an overhead transparency, a 35mm slide, or a "conventional" print.

- EXPORTING A SLIDE

Your slide can be "exported" to another terminal via the VAX/VMS mail utility, to any other user having a DECslide-supported terminal. Receiving users can see the slide just as it appeared on your own terminal, then can store, print, or delete it.

- MANAGING THE LIBRARY OF SAVED SLIDES

You can also "export" any finished slide to a library-like utility called the Slide Organizer. As a feature that also works for DECgraph output, the slide organizer can store your slides in one or more "slide trays," enabling them to be printed sequentially in a batch job or arranged for presentations to be given.


1
2
3
4
5
Y TRAINING COURSES
ES SCHEDULES JOBS
STINGS BULLETIN BOARDS
NEWSLETTERS PRICE LIST
OS REGISTRATION
DISPLAYS DICTIONARIES
ITIES PLANT SECURITY
RGENCY TELEPHONE DIRECTORY
CEDURE MANUALS CATALOGS
ANDARDS COMPETITIVE DATA
ORY AND NEW SERVICES
CHART MEMBERSHIP LISTS
PROJECT MANAGEMENT REPORTS

Chapter 11 ▪ VAX VTX

VAX VTX is a videotex system designed for internal use by businesses and other private organizations. It is a VAX/VMS software product which conforms to international standards for videotex systems; it requires no specialized hardware. VAX VTX delivers information quickly and simply to a variety of computer terminals and personal computers. It can be integrated into Digital's ALL-IN-1 office automation system without any additional hardware. It can also be added as an extremely easy-to-use front end to an organization's transaction processing applications without requiring major rewriting of existing systems.

▪ Who Uses VAX VTX

Organizations that want to present some or all of their internal publications and transaction processing applications simply, uniformly, and efficiently will benefit from VAX VTX. Whether the organization's information is on a single computer or distributed worldwide over a computer network, VAX VTX offers fast, up-to-date information to users.

Using VAX VTX, a department head can make frequently requested and frequently modified departmental publications available from a single source, make them easy to keep current, and keep them readily available for widespread use. Employees who wants quick, simple, reliable access to a variety of information (applications ranging from weather reports, airline reservations, and motel information to personnel policies and procedures, sales information, directories, employee activities, credit union transactions) have a convenient source that saves time and effort.

VAX VTX runs under the VMS and MicroVMS operating systems and requires DECnet-VAX software as a prerequisite. It can be used on the MicroVAX I and all other VAX systems.

▪ Benefits of VAX VTX

▪ The VAX VTX end user documentation consists of one small card. The 18 functions available to you to use VAX VTX are truly simple to learn. The actual subscriber's card is illustrated below.

VAX VTX
DEC terminals card

Function	Key Sequence
HELP	PF2
PAGE HELP	PF1 – PF2
MAIN	PF1 – PF3
BACKUP	PF3
NEXT PAGE	PF4
PREV PAGE	PF1 – PF4
EXIT	PF1 – .
SET MARK	PF1 – 8
GOTO MARK	PF1 – 9
FIND	PF1 – 7
REVEAL	PF1 – 4
LOCAL	PF1 – 5
ENTER	ENTER
FORM ENTER	PF1 – ENTER
REFRESH	CTRL/W
CLEAR CHAR	DELETE
CLEAR ENTRY	CTRL/U
GET AGAIN	PF1 – 0

HELP	Displays general information about using VAX VTX.
PAGE HELP	Displays help information about the current page.
MAIN	Displays the first page you saw when you started VTX.
BACKUP	Displays the previous menu page.
NEXT PAGE	Displays the next page in a series of continuation pages.
PREV PAGE	Displays the previous page in a series of continuation pages.
EXIT	Gets you out of the VTX service.
SET MARK	Marks a page for future reference. Use numbers 0-4 for bookmarks.
GOTO MARK	Displays a previously marked page whose bookmark you specify.
FIND	Displays the page whose keyword or page number you specify.
REVEAL	Displays control information (page number, keywords, etc.) for the current page.
LOCAL	Lets you issue the local VTX commands PRINT, SAVE, and EXIT. For a complete explanation of local commands use the HELP function.
ENTER	Sends information, such as a menu selection, from your terminal to VTX.
FORM ENTER	Sends all information on a form page to VTX.
REFRESH	Displays the current page again, eliminating any line interference that might have occurred the first time you displayed the page.
GET AGAIN	Retrieves and displays the current page again with any changes that have been made to the page since you last retrieved it. If you use the GET AGAIN function while working on a form page, you will get a clean version of the form page.

Figure 11-1 ▪ VAX VTX Subscriber's Card

VAX VTX does not require the use of any special editor. This means that information may be prepared using any standard Digital word processing, text editing, forms management, or graphics products. Users can use such products as WPS, EDT, PECO, DECgraph, DECslide, VAX TDMS, or VAX FMS to create text and graphics for the information base. VAX VTX can also be integrated into Digital's ALL-IN-1 office menu system to offer true office automation along with videotex information distribution and retrieval.

- VTX is easy to use on a variety of terminals. The subscriber can access all of the VAX VTX functions from the terminal keypad of a variety of industry standard terminals. VAX VTX works with VT200-and VT100-compatible terminals, personal computers, and workstations including the DECmate, Rainbow and Professional workstations. It also supports the IBM* PC running the CROSSTALK XVI* VT100 emulation software. Retrieving pages from a VTX information base is a simple matter of selecting items from menus.

- VAX VTX is designed to allow easy distribution of information and efficient transmission of remote information to any node in a distributed VTX system. For example, a VAX VTX user in the United States might look at a VTX menu page that offers selections including: 1. European Sales Information and 2. North American Sales Information. If the subscriber selects choice 1, VTX might actually establish a DECnet link to a system in England and start retrieving information from the information base there. A user in England might see the same menu and select choice 2, thus establishing a remote link to the computer in the United States. When the pages in the information base are designed according to the same standards, users won't even suspect that they originated in different locations. The advantage of this distributed capability is that the information is equally accessible to all users on the VAX VTX system, yet can be located on the computer closest to the people who supply the information.

- VAX VTX supports the transmission of ASCII, ReGIS, Prestel, and NAPLPS pages provided the subscribers have the appropriate hardware to display the protocol of requested pages. Standard VAX VTX currently supports all standard Prestel terminals as well as the NAPLPS display-protocol on Digital's Professional workstation running PRO/NAPLPS software, the Sony VDX-1000* videotex unit, and the Norpak Corporation MARK IV* videotex decoder. This protocol independence means existing VMS files, or any files that can be converted to a VMS format, can be adapted into VAX VTX pages. In addition, because VAX VTX is a delivery system that is not restricted to or limited by particular display protocols, it is possible to design a single system that supports more than one display protocol simultaneously.

- The VTX Information Provider's Assistance Tool (IPAT) enables information providers to work on numerous pages in an information base without interfering with the performance of the VTX service. With IPAT, individuals with only a few days training can specify control information such as page number, security code, keywords, and expiration date for pages they want to enter in a VTX information base. Other formatting capabilities, like linking the display portion of a VTX page with the page's control information and preparing batches of page action requests for use by the VTX Update Utility to include them in a VTX information base, are also performed using the IPAT.

- A VTX information base is modified when a VTX operator or information provider submits an update command file to the VTX Update Utility. The update command file can contain one or more page action requests to add, change, or remove a information base page. Because of this method of information base updating, it is easy for a VTX site to make most of its information base updates during times when system usage is low.
- The VTX Account Control Utility (VTXACU) allows the VTX operator to establish accounts for VTX users, specify a language and password for each user, and assign users to closed user groups, thus determining who can retrieve what. This means different levels of security protect sensitive information without denying access to casual users.

▪ VAX VTX — A Closer Look

Managing a VAX VTX system involves:

- Designing an information base
- Using the VTX Information Provider's Assistance Tool to build pages for inclusion in the information base
- Using the VTX Update Utility to populate and maintain the information base
- Using the VTX Account Control Utility (VTXACU) to establish and maintain user accounts
- Using the VTX Control utilities to bring up the information base daily

Designing an Information Base

A VTX information base consists of pages of information logically organized as a hierarchy. To design an effective VTX information base, you should understand how subscribers navigate through a hierarchy of pages. A VTX information base branches from a single source, the main menu page, to various more specific menu pages, each of which leads eventually to pages of information. Subscribers navigate from general menu pages to more specific menu pages, to informational pages, and back to more general menus by selecting menu choices and by using VTX keypad functions like BACKUP and MAIN MENU.

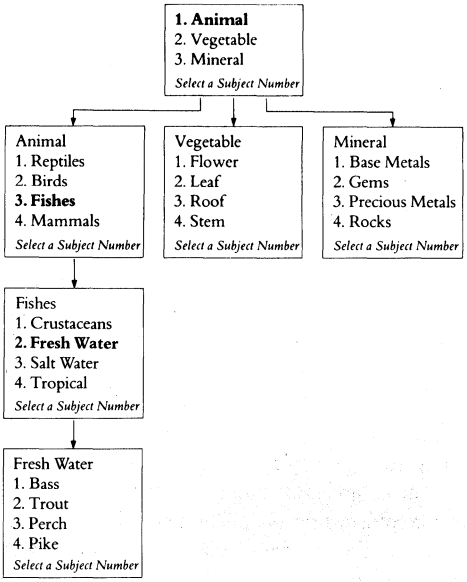


Figure 11-2 ▪ Sample Tree Structure of a Videotex Database

Menu pages are the signposts in a VTX information base which point the way to pages that subscribers might want to retrieve. Not only are menu pages useful to subscribers navigating through an information base, they are useful to the information base designer who wants to create a logical outline for a VTX information base. If you design your VTX information base by thinking of categories and subcategories of information, you can easily represent the logical structure with menu pages alone. You can assign certain page ranges to certain categories of pages in your information base. You can subdivide your information base on the basis of:

- subject – probably the most common basis for categorizing information
- security – some categories can be made available to all subscribers while access to other categories can be restricted.
- language – VAX VTX allows you to offer multiple information bases through the same service. For example, an organization with locations in two countries might want to create two information bases with identical information in two different languages.
- protocol – VAX VTX supports multiple protocols, so you may want to create two, one for NAPLPS pages and one for ASCII pages. The page ranges can be assigned accordingly.

Building Pages in a VAX VTX Information Base

Once you have determined the general structure of your VTX information base, the various information providers can start building pages. Using IPAT, the information providers can easily create both menu pages and display pages in whatever order they prefer. The display portion of a page is created with any software editor the information provider chooses.

To supply control information for a menu page, an information provider would also use IPAT.

Menu page editing is equally straightforward. When an information provider finishes using the page editing screen, the IPAT Menu Editing Screen appears at the terminal. On this screen, the information provider matches each menu choice from the initial menu with the page number of the database page to which it points. Choice number 2, for example, could point to page 01232.

Once the information provider indicates that the information on the menu editing screen is complete, the display and control portions of the page can be merged in an update command file and submitted to the VTX Update Utility for inclusion in the information base. The information provider can end the IPAT session now or continue to create new pages and submit them all at once. The information provider might, for example, prefer to create all the pages that branch from the initial menu page before ending the IPAT session.

Populating and Maintaining a VTX Information Base

Populating a VTX information base is simply the process of adding pages to it. Maintaining the information base involves both adding and deleting pages. Generally a VTX operator populates and maintains an information base by submitting information providers' update command files to the VTX Update Utility. The update command files contain one or more page action requests specifying information about pages to be added or deleted.

Maintaining VTX User Accounts

With the VTX Account Control Utility (ACU), a VTX system operator can easily create, modify, and delete user accounts. A system operator can also display account information and reset the usage statistics in one or all user accounts. The ACU is useful for establishing security for the information base on the basis of closed user groups. In other words, a VTX operator can group subscribers by security clearance then pass the information on to information providers so that they can assign the proper group codes to the pages they create.

As an example of how to use the ACU utility, suppose you displayed the account information for a user named Peters and saw that he had been assigned to closed user group 3. If, for some reason, Peters needs access to more of the VTX information base, say those pages available only to members of closed user group 4, the operator can easily modify his security clearance with just a few commands.

Bringing Up the VTX Information Base

VTX comes with a set of control utilities that enable the VTX operator to bring up a VTX service every day and make it available to subscribers. Bringing up VTX involves:

- Starting the VTX information base server process.
  The server is like a waiter in a restaurant and is largely responsible for the efficient transmission of information that characterizes a VAX VTX service. A server receives subscribers' requests for pages in the information base, retrieves the pages, and sends the pages back to the requesting terminals where they are formatted and displayed according to the characteristics of the pages and the capabilities of the terminals. Because a server is a single process and does not concern itself with formatting pages at subscribers' terminals, the server makes very efficient use of system resources while handling a large number of simultaneous requests from subscribers. The alternative to a single server handling multiple requests is for each subscriber to run a VMS process, thus quickly crowding the operating system in the same way restaurant patrons would crowd the kitchen if there was no waiter to serve them.
- Enabling the server to receive and transmit information from subscribers and the information base.
- Opening the information base files to make their pages available to subscribers.
- Declaring any remote servers or applications so that the local server knows where to find them when a subscriber selects a menu choice that points to a remote process. Declaring a remote process is like posting a phone number to which the server can refer. Think again of the restaurant analogy. If a seafood restaurant wants to offer pizza on its menu and have the pizza cooked by the pizza shop across the street, a patron does not have to know where the pizza comes from. As long as the pizza shop's phone number is posted in the kitchen, the waiter can call across the street when a patron orders pizza. In this analogy, the pizza shop is the remote server. Posting the pizza shop's phone number in the kitchen is analogous to declaring the pizza shop as a remote server.

Glossary

%ALL:
The parameter to a TDMS INPUT, OUTPUT, or RETURN request instruction that permits you to map data between all identically named form and record fields without specifying the individual fields. If the Request Definition Utility finds an error in an INPUT %ALL, OUTPUT %ALL, or RETURN %ALL mapping, it does not include that mapping when it stores the request in the CDD. At run time, TDMS executes only the correct mappings.

See also implicit mapping.

Access Control List (ACL):
A table that lists which users are allowed access to an object, and what kind of access. The CDD maintains ACLs for DATATRIEVE, DBMS, and TDMS. ACMS and Rdb/VMS maintain their own ACLs.

- In CDD, ACLs allow or deny users access to protected CDD directories or objects. Each dictionary directory, subdictionary, and object has an access control list associated with it. You can attach ACLs to such dictionary objects as:
  - DATATRIEVE domain and procedure definitions
  - TDMS requests and request library definitions
  - DBMS schema, subschema, and storage schema definitions
  - CDD record definitions

  Each product provides syntax for defining the ACL to be attached to its elements. The easiest way to define ACLs is through the CDD Dictionary Management Utility. DMU provides an ACL editor, which lets you change an ACL by editing a screen display.
- In ACMS application definitions, the ACL is the task control characteristic that determines who can select a task. You use the ACMS Application Definition Utility to define the ACL for a task.
- In Rdb/VMS, ACLs allow or deny users access to databases, relations, and views. You can explicitly define the access rights or copy them from an existing database, relation, or view, either from within the same database or from another database. The ACL for a database and each of its elements is stored in the database file.

See also privilege.

Access Mode:
A characteristic of a transaction that describes what kind of operation you intend to perform on data in a database.

- In the DBMS Data Manipulation Language, you use the READY statement's USAGE clause to specify RETRIEVAL ("read only") or UPDATE ("read and write").
- In the Rdb/VMS START_TRANSACTION statement, you specify READ_ONLY or READ_WRITE.

See also usage mode and allow mode.

ACL:
See access control list.

ACMS:
See Application Control and Management System.

ACMS Central Controller:
The ACMS process that serves as the central control point for the ACMS run-time system. The ACMS/START SYSTEM command starts the ACMS central controller, which starts up and controls the other ACMS run-time components.

ACMSGEN:
The utility used to set ACMS system parameters. Similar to the VAX/VMS SYSGEN Utility.

ACMS Operator:
An ACMS user authorized to control the daily operations of ACMS and/or its parts with the ACMS Operator commands.

ACMS Operator Command:
One of several DCL commands provided by ACMS to control the operations of the ACMS system software and ACMS applications. Many ACMS Operator commands require the VMS OPER privilege.

ACMS User:
A VMS user authorized to access or control ACMS or its parts.

Active Form:
The TDMS form, referenced in a request, that is used during a single programming request call. A conditional request can reference more than one form, but only one form can be active at any one time.

ADB:
See application database.

ADT:
See Application Design Tool.

ADU:
See Application Definition Utility.

After-Image Journal:
A file that contains images of records after they have been updated. You can use the After-Image Journal to reconstruct a restored database up to the last successfully completed transaction. After-image journaling is also called long-term journaling.

Agent:
A VMS process through which one or more terminal users access the ACMS runtime system. All terminal users submit tasks through an agent. ACMS provides one agent, called the command process, that acts for all terminal users.

See also command process.

Aggregate Expression:
See statistical expression.

Allow Mode:
A characteristic of a transaction that describes the level of protection that you will provide for the data you want to work with.

- In DBMS, the allow mode is part of the READY statement's USAGE clause in the DBMS Data Manipulation Language. The allow mode can be BATCH, CONCURRENT, PROTECTED, or EXCLUSIVE.
- In the Rdb/VMS START_TRANSACTION statement, you specify EXCLUSIVE, PROTECTED, or SHARED.

See also usage mode and access mode.

Ancestor:
With respect to a CDD directory or object, a preceding dictionary or subdictionary directory in the CDD hierarchy. Ancestors have as descendants all related dictionary directories and objects that follow them in the hierarchy.

Application:

- A logically related set of data processing operations that support a particular business activity.
- In ACMS, a set of tasks that are related in terms of the business activity they support and that are controlled as a single unit. An ACMS application is defined with the ACMS Application Definition Utility (ADU) and runs under the control of the ACMS run-time system. An application definition specifies operational characteristics for the tasks and servers of the task groups that make up the application.

Application Control and Management System (ACMS):
A software product, layered on VAX/VMS, used to reduce the lifecycle costs involved in designing, developing, maintaining and controlling transaction processing and other complex VAX/VMS applications.

Application Database (ADB):
A run-time database that contains information derived from application and task group definitions. An application database is generated by building an application definition with the Application Definition Utility (ADU). The ACMS run-time system uses application databases to determine what processes to start, when to start them, and which users have access to which tasks.

Application Definition Utility (ADU):
The primary tool for creating ACMS applications. The Application Definition Utility provides the commands and clauses for defining tasks, task groups, applications, and menus.

Application Design Tool (ADT):
A DATATRIEVE utility that aids you in creating domains, record definitions, and files by prompting you with questions at each step in the process.

Application Designer:
A system user, often referred to as a system analyst, responsible for the overall specification and definition of an application and its parts.

Application Execution Controller:
The ACMS component that controls task execution for all the tasks in an application. Each application has its own application execution controller. Application execution controllers start up and control the server processes needed to handle processing work for tasks. They also handle exchange steps, step actions, and the sequencing of steps for tasks defined with ACMS. Application execution controllers reference application databases, task databases, request libraries, and message files.

Application Manager:
A system user responsible for the overall operation and maintenance of an application.

Application Program:
A sequence of instructions and routines, not part of the basic operating system, designed to serve the specific needs of a user. An application program can use a database system to access data.

See also run unit.

Application Programmer:
A system user responsible for creating and maintaining part or all of the programs for an application. The application programmer generally works in a high-level language (such as COBOL or BASIC) to implement the design established by the application designer. In some organizations, a single individual does the work of an application designer and application programmer.

Application Programming Services:
A set of subroutines, provided as part of VAX ACMS, that can be called from an application program.

Area:
In DBMS, a subdivision of the database, named in the schema, that corresponds to an RMS file. Any number of record types can be stored within an area. One or more areas make up a subschema realm.

See also realm.

Array:
A data structure consisting of more than one element, in which all elements have the same characteristics and are referenced by the same variable name.

In a TDMS form or record, an array is a field that contains several elements referenced in a request by the same name and having the same characteristics (length, data type, and so on).

Ascending Order:
An order of sorting that starts with the lowest value of a key and proceeds to the highest value, in accordance with the rules for comparing data items.

See also sort key, descending order

Asynchronous Call:
A call to a TDMS subroutine that begins, but does not necessarily complete, the requested operation before allowing your program to continue execution. At some later time, the requested operation will complete and notify the program. In the meantime, your program and the requested operation can both proceed at the same time.

See also synchronous call.

Attribute:
See field.

Audit Trail:
In ACMS, a monitoring tool that has a recording facility and a utility for generating reports. The recording facility gathers information about a running ACMS system and writes the information in the Audit Trail Log file. The information in the Audit Trail Log includes system and application starts and stops, user logins and logouts, processing errors, user task selections, task completions, and task cancels. The report utility can generate summary reports of the information in the log file.

In CDD, a collection of the history list entries for a dictionary directory, subdictionary, or object, created with the /AUDIT qualifier.

See also history list.

AUTOMATIC Member:
In DBMS, a record that is automatically inserted into a specified set when the record is stored in the database. AUTOMATIC set membership is specified in the schema.

Bachman Diagram:
In DBMS, a graphic representation of the set relationships between owner and member records used to analyze and document a database design.

Background Text:
The text on a run-time TDMS form that is displayed whenever the form is displayed. For example, EMPLOYEE NAME:_____.

Bar Chart:
A chart that uses vertical rectangles (bars) to show the relationship between values on the X and Y axes. The height of the bar signals the Y value. Except for the histogram, the bars are of equal width. (In a histogram, the bars may be equal or unequal in width, depending on the data.)

See also clustered bar chart, histogram, stacked bar chart.

Batch Processing:
A mode of computer operation in which the commands and data that control the actions of the computer are entered by a programmed script rather than a person at a terminal.

Before-image Journal:
A file that contains images of records before they have been updated. DBMS and Rdb/VMS use before-image journaling to automatically undo updates to a database when a transaction is rolled back. Before-image journaling is also called recovery-unit journaling or short-term journaling.

Block Step:
One of three kinds of steps used to define the work of a multiple-step ACMS task. A block step has three parts: attributes, work, and action. The work part of a step block contains a sequence of exchange and processing steps.

Boolean Expression:
A string of symbols that specifies a condition that is either true or false. For example:

PRINT PERSONNEL WITH STATUS = 'TRAINEE' AND AGE LT 30

Here, the Boolean expression is STATUS = 'TRAINEE' AND AGE LT 30. The PRINT statement displays only those records for which the value of this expression is "true".

See also Boolean operators, conditional expression.

Boolean Operators:
Symbols or words that enable you to join two or more Boolean expressions. Boolean operators are AND, OR, and NOT. For example, the expression STATUS = 'TRAINEE' AND SALARY > 10000 contains the Boolean operator AND.

Border:
In DECgraph, lines that enclose a chart. You can choose a full border, which gives a four-sided box; a corner border, which gives lines on the left and bottom sides of the chart; or no border.

Broadcast Message:
A text line that lets you know of an event outside the DECslide environment, such as notice of mail or of a system shutdown.

See also message flag.

Build Operation:
The execution of the BUILD LIBRARY command in the TDMS Request Definition Utility (RDU). This operation places in a VAX/VMS request library file the requests named in the request library definition and their associated form and record information. The program accesses this file at run time to execute a request.

The build operation is successful if RDU finds that all of the requests, forms, and records exist and that the mappings in the requests are correct. As a result, RDU creates a file in your default directory with an .RLB file type.

CALC Mode:
In DBMS, a way to calculate a record's storage address in the database by using the value of one or more data items in the record. CALC mode is declared in the storage schema and can be used only with SYSTEM-owned sets.

Call Interface:
A mechanism for a program to access components of a software product. For example, the VAX DATATRIEVE Call Interface is the part of DATATRIEVE that provides access to DATATRIEVE's data management services. There are three modes of access to DATATRIEVE's call interface:

- Through the Terminal Server
- Through the Remote Server
- From a calling program

The call interface for DECslide consists of a single entry point, SLIDE$COMMAND, to which you pass a DCL SLIDE command string. To call DECgraph, you pass a DCL GRAPH command string to the GRAPH$COMMAND entry point.

See also program interface.

Callable DBQ:
In DBMS, a data manipulation interface to the Database Control System that allows programs written in any VAX language that conforms to the VAX Calling Standard to access a database.

See also database query and Interactive DBQ.

Callable Interface:
See call interface.

Callable RDO:
A single external routine that accepts an Rdb/VMS statement as a parameter. You can call this routine from any language that adheres to the VAX Procedure Calling Standard. Callable RDO allows a program to u se Rdb/VMS even if no precompiler exists for the language.

Calling Program:
A program that issues calls to other programs or subprograms to execute certain operations.

In VAX DATATRIEVE, for example, a high-level language program that contains calls to callable DATATRIEVE routines is referred to as the calling program.

Cancel Action:
A procedure or image called by an ACMS task when the task is canceled. The cancel action does cleanup work for the task, such as recovering from incomplete operations; it does not release locks or perform other work specific to a server.

Cancel Procedure:
A procedure called by an ACMS task when the task is canceled if, at the time of the cancel, the task is processing in or keeping server context in a procedure server process. The cancel procedure does cleanup work for the server process, such as releasing record locks, so that the process can be reused without being restarted. When a cancel procedure is called, it runs in the server process allocated to the task, whether or not the task is using the server process at the time of the cancel.

Candidate Key:
A field or set of fields that uniquely identifies the individual records of a relation. For example, in a relation of employee information the employee identification number is a candidate key.

Cardinality (of a relation):
The number of records in the relation.

Cartesian Product:
See cross product.

Case Value:
A literal string in a TDMS request that determines whether a conditional request instruction executes at run time. TDMS checks that the case value matches the value in a control field. If there is a match, TDMS executes the request instructions associated with the case value.

CDD:
See Common Data Dictionary.

CDDL:
See Data Definition Language Utility.

CDDL Source File:
A file in which you define CDD records. The CDDL compiler inserts these definitions into the CDD directory hierarchy.

CDDV:
See Dictionary Verify/Fix Utility.

CHAIN Mode:
In DBMS, a way to link records sequentially using NEXT, PRIOR and OWNER pointers. CHAIN mode is declared in the storage schema and cannot be used with sorted sets or CALC sets.

Change Menus:
Two word list menus — one for text and the other for graphics objects — with options for size changes, rotation, patterns, italic text, and text editing.

Character String:
A string of characters (bytes) that is identified by an address and a length and that is used in the callable interface to pass the SLIDE command string to DECslide.

Chart:
A graphic presentation of data. DECgraph provides bar, line, pie, and scatter charts.

Child:
A way of describing a dictionary directory, subdictionary, or object in the CDD that immediately succeeds another directory or subdictionary in the CDD hierarchy. The preceding directory or subdictionary is called the parent. Each dictionary child has precisely one parent. For example, given CDD$TOP and CDD$TOP.MANUFACTURING, CDD$TOP is the parent and CDD$TOP.MANUFACTURING is the child.

See also descendant, parent, and ancestor.

Clustered Bar Chart:
A chart that consists of groups of vertical bars placed side by side with space between each group of bars.

CLUSTERED VIA Set Option:
In DBMS, a record placement option in which the Database Control System (DBCS) stores records on or near the page that contains the owner of the set. The CLUSTERED VIA option is declared in the storage schema.

CODASYL:
An acronym for the Conference on Data Systems Languages, the committee that designed the COBOL language and provided the guidelines used in the development of VAX DBMS.

CODASYL-compliant:
Any database system that conforms to the guidelines set by the Conference on Data Systems Languages.

Collating Sequence:
The sequence in which characters are ordered for sorting, merging, and comparing.

Collection:

- In VAX DATATRIEVE, a type of record stream formed with the FIND statement. You can name a collection in order to have several collections available at once.

- In DBMS, all occurrences of records that belong to a specific record type. Record types are defined in schema and subschema entries.

See also CURRENT.

Column:
A set of numbers or words arranged vertically. For example, the DECgraph Keyboard Data Entry Screen has a column of Y legend values ranging from Y1 (bottom) to Y6 (top).

See also field.

Column Headers:
The heading that labels the columns of data in a DATATRIEVE report or in the output of a DATATRIEVE PRINT statement.

Command:
An instruction, generally an English word, typed by the user at a terminal or included in a command procedure that requests the software monitoring a terminal or reading a command procedure to perform some predefined operation.

In DATATRIEVE, a command is distinct from a statement. DATATRIEVE commands usually deal with the Common Data Dictionary and perform data description functions. DATATRIEVE commands cannot be combined with each other and cannot be used in statements.

See also statement.

Command Line:
See command string.

Command Process:
The process in the ACMS terminal control subsystem that handles user login and interaction between terminals and ACMS.

Command String:
A line (or set of continued lines) consisting of a command and, optionally, information modifying the command, including its qualifiers and parameters. For example, in DECslide, the SLIDE command string uses the following format:

SLIDE[/qualifiers] [slide_name] [new_slide_name]

In DECslide and DECgraph, you can use the command string for either an interactive or noninteractive session.

See also interactive mode, noninteractive mode.

Comment Character:
In DECgraph, either a C, an asterisk, or exclamation point used to indicate a comment in the load file. Comments let you document the contents of your load file; they do not appear as data. For example:

```
C The data type for X values is TEXT.
X_DATA_TYPE TEXT
```

COMMIT:

- In DBMS, a function that terminates a recovery unit and makes permanent all database updates initiated by the recovery unit.
- In ACMS, also an Application Definition Utility keyword used when defining tasks with recovery.
- In Rdb/VMS, a statement that completes a transaction by entering the changes in the physical database file.

See also RETAINING and rollback.

Common Data Dictionary (CDD):
A central storage facility consisting of a hierarchy of directories that contain definitions used by VAX Information Architecture products. The CDD contains descriptions of data, not the data itself. CDD objects are stored hierarchically and are accessed by reference to dictionary path names.

CDD directories and subdictionaries contain objects such as:

- ACMS application, menu, and task group definitions
- DATATRIEVE domain, record, and procedure definitions
- DBMS schema, subschema, and storage schema definitions
- TDMS record and form definitions, requests, and request library definitions
- Rdb/VMS database, relation, field, index, and constraint definitions

Compound Object:
An object formed by using the JOIN key to combine more than one graphics object, text object, or both.

COMPUTED BY Fields:
Virtual fields that appear in a DATATRIEVE record definition or an Rdb/VMS relation or view definition, but not in the physical record. Because the value of a COMPUTED BY field is computed as part of a statement, it occupies no space in the record.

Concurrency:
The simultaneous use of a database by more than one user.

Conditional Expression:
A string of symbols that can be evaluated as true or false. For example, in the statement FOR E IN EMPLOYEES WITH E.STATE = "MA", the conditional expression is E.STATE = "MA". Also called Boolean expression.

See also Boolean expression.

Conditional Instruction:
A TDMS request instruction that executes only if certain conditions are true. TDMS executes a conditional instruction if the value in a control field matches the case value specified within the conditional instruction.

Conditional Request:
A request containing one or more conditional instructions.

See also conditional instruction.

Constraint:
A set of criteria that restricts the values in a field. In Rdb/VMS, you set up constraints using the DEFINE CONSTRAINT statement.

Context Variable:
A temporary name that identifies a record stream to Rdb/VMS.

Once you have associated a context variable with a relation, you use only that context variable to refer to records from that relation in the record stream or loop you created. In this example, E is a context variable:

```
FOR E IN EMPLOYEES PRINT E. END_FOR
```

You can also use context variables in DATATRIEVE to resolve context ambiguity.

Control Field:
A program record field, also specified in a TDMS request, whose value determines whether or not TDMS executes a conditional instruction.

See also conditional instruction.

COPY Field Description Statement:
A CDDL statement that inserts the field descriptions of existing records into the descriptions of new records.

Cross Operation:
See join operation.

Cross Product:
A relation that is the result of performing a join operation to combine every row in one relation with every row in another.

Currency Indicators:
DBMS pointers that serve as place markers in the database for the Database Control System (DBCS) and your run unit.

CURRENT:

- In DBMS, identifies which database records or positions are being used as currency indicators.
- In DATATRIEVE, identifies the most recently formed collection.

Current Slide:
The slide currently in the work area during design mode.

Cursor:
A highlighted area that indicates your place on the screen. In DECslide, the cursor for graphics objects is diamond shaped. The text cursor is rectangular.

Data Definition Language Utility (CDDL):
The VAX CDD utility that lets you insert record definitions into the CDD. You create the data descriptions in a CDDL source file, and you compile the source file with the CDDL compiler.

Data Definition Languages (DDL):
In VAX DBMS, the languages used to describe schemas, subschemas, and storage schemas.

In VAX Rdb/VMS, a set of statements that allow you to define the structure and characteristics of stored data. You use data definition language to describe fields, relations, views, indexes, and constraints. The Rdb/VMS data definition language is part of RDO, the interactive Rdb/VMS utility.

See also schema data definition entry, storage schema data definition entry, and subschema data definition entry.

Data Entry Phase:
The stage in which you provide data to DECgraph through the Keyboard Data Entry Screen, a load file, or VAX DATATRIEVE. The DATA icon leads to a submenu of icons representing the three methods for entering data.

Data File:
A DECgraph file that contains the actual data for a chart, including the X and Y values. Data files have the file type _.GRD.

Data Item:

- In DBMS, the smallest unit of named data in a record type. A data item can be a single value or an array of one or more values.
- In Rdb/VMS, the smallest unit of data. A data item occupies a single field in a record.

Data Item Occurrence:
In DBMS, one occurrence of a data item type.

See also data item type and record occurrence.

Data Item Type:
In DBMS, the smallest unit of defined data. A data item type can represent a single value or an array of one or more values.

See also data item occurrence and record type.

Data Manipulation Facility:
The part of DATATRIEVE that parses, optimizes, and executes all commands and statements passed to DATATRIEVE.

Data Manipulation Language (DML):
In DBMS, the statements that permit programs written in VAX languages to access the database.

In Rdb/VMS, A set of statements that allow you to store, retrieve, modify, and erase data from a database. Rdb/VMS provides two methods of manipulating data:

- Embed the data manipulation statements in a high-level language such as COBOL.
- Issue the data manipulation statements directly, using the RDO utility.

Data Set:
One file that includes both the identification and data files, all the basic information needed to produce a chart.

See also data file, identification file.

Data Type:
A characteristic assigned to a field that determines the kind of data the field can contain.

In TDMS, you determine the data type of a form field in the field identifiers and field validators of the form definition. You determine the data type of a record field in the data type statement of a record definition.

Data Value:
A user-assigned value of a data item occurrence.

See also data item occurrence and data item type.

Database :
A collection of interrelated data on one or more mass storage devices. The collection is organized to facilitate efficient and accurate inquiry and update.

In a database, more than one user can access the data at the same time. Data integrity and security are provided by the database.

See also relational database.

Database Administrator (DBA):
The person or group of people responsible for planning, designing, implementing, and maintaining a database.

Database Control System (DBCS):
The DBMS or Rdb/VMS component that, together with the VAX/VMS operating system, provides run-time control of database processing.

Database Key (dbkey):
In DBMS and Rdb/VMS, a unique value that identifies a record in a database. The Database Control System assigns the value when a record is stored in the database.

In DBMS, your run unit cannot directly access database keys, but they are used by the Database Control System whenever you store, retrieve, or manipulate a record.

In Rdb/VMS, your program can retrieve the database key and use it to access a record.

Database Management System:
A system for creating, maintaining, and accessing a collection of interrelated data records that may be processed by one or more applications without regard to physical storage. Data is described independently of application programs, providing ease in application development, data security, and data visibility.

The VAX Information Architecture includes two database management systems:

- VAX DBMS, a DIGITAL software product that complies with the standards for database management systems established by CODASYL
- Rdb/VMS, a database management system based on the relational data model

See also database, Rdb/VMS, and relational database.

Database Pages:
The structures used to store and locate data in a DBMS or Rdb/VMS database. Database pages consist of one or more disk blocks of 512 bytes each.

DBMS uses page-clustered I/O, a technique that retrieves groups of physically-related database pages, rather than an individual page, in response to a run unit's request for data.

Database Query (DBQ):
In DBMS, a data manipulation utility that interprets data manipulation statements. DBQ provides access to data through both interactive and callable modes. Interactive DBQ is a DBMS query language. Callable DBQ provides access to the database for programs written in high-level languages.

See also Callable DBQ and Interactive DBQ.

DATATRIEVE:
A VAX data management language for manipulating, storing, and modifying records from RMS data files, DBMS databases, and Rdb/VMS databases. DATATRIEVE is callable from a variety of high-level languages.

DATATRIEVE Procedure:
See procedure.

DBA:
See database administrator.

DBCS:
See Database Control System.

Dbkey:
See database key.

DBMS:
Used commonly, DBMS can refer to any database management system. In VAX Information Architecture documentation, DBMS usually refers to VAX DBMS, a DIGITAL software product that complies with the standards for database management systems established by CODASYL.

See also database management system.

DBQ:
See database query and Callable DBQ.

DBR:
In DBMS, the name of the process that performs database recovery. It is called by the Monitor at restart.

DCL:
DIGITAL Command Language.

DCL Command Procedure:
A sequence of DIGITAL Command Language (DCL) commands stored in a file; sometimes referred to as a DCL procedure.

DCL Server:
One of two types of servers used to handle processing work for ACMS tasks. A DCL server handles images, DATATRIEVE commands, and DCL commands and command procedures.

See also server, DCL server image, and procedure server.

DCL Server Image:
The image, provided by ACMS, that is loaded into a DCL server process when the process is started by the application execution controller. The DCL server images allows you to use images, DATATRIEVE commands, and DCL commands and procedures to implement processing for tasks.

See also procedure server image.

DCL Server Process:
See server process.

DDL:
See data definition languages.

Deadlock:
A situation in which two or more processes request the same set of resources and there is no method for resolving the conflict. For example, if process A has record 1 locked and requests record 2 while process B has record 2 locked and is requesting record 1, a deadlock occurs between processes A and B.

VAX DBMS resolves all deadlock situations.

Deadly Embrace:
See deadlock.

DECnet:
The DIGITAL software facility that enables a user to access information on a remote computer via telecommunications lines. DECnet/VAX enables a VAX/VMS operating system to function as a network node.

Default:
A value that is assumed unless — or until — you specifically indicate another choice. For example, Palette 1 is the default palette for DECslide and DECgraph.

Default Dictionary Directory:
The CDD directory assigned to you when you invoke an image that uses the CDD. This directory becomes the starting directory for path names. You can define a directory as the default by assigning a path name to the VAX/VMS logical name CDD$DEFAULT. If you do not, the default directory is CDD$TOP. The CDD Dictionary Management Utility and some command qualifiers allow you to set temporary default directories. You can also set the default directory with the DATATRIEVE SET DICTIONARY command.

Default Directory:
In DECslide, the storage area used to locate the files where you are creating your slide designs. You always save your slides in your default directory.

See also slide directory.

Default Slide:
A slide DECslide identifies with the name CURRENT that you can use instead of specifying a slide name in answer to a file option prompt. You can change this name for a DECslide session when you specify the input and output parameters in the command string.

Degree (of a relation):
The number of fields in a relation definition.

Delete Access:
File protection that you set to let a user delete a file from your VMS directory. This protection must be set, for example, when you use DECslide's delete option and specify another user's slide.

Descendant:
A way of describing a dictionary directory, subdictionary, or object in the CDD that follows another directory or subdictionary in the CDD hierarchy. A dictionary directory or subdictionary owns all its descendants. CDD$TOP owns all the dictionary directories, subdictionaries, and objects, and they are all descendants of CDD$TOP.

See also child and parent.

Descending Order:
An order of sorting that starts with the highest value of a key and proceeds to the lowest value, in accordance with the rules for comparing data items.

Descriptor:
A data structure that specifies the address, length, and data type of a string.

Design Mode:
The mode of operation that lets you use DECslide's keypad keys and design your slides. You enter design mode when you select the DESIGN icon.

Design Phase:
The DECgraph stage in which you identify the type of chart you want and its particular design features, such as colors, borders, grids, palette, range, scale, and so forth. You begin the Design phase by selecting the DESIGN icon.

Detail Lines:
The formatted data lines in a DATATRIEVE report or PRINT statement.

Device Definition Utility:
The ACMS tool for defining which terminals have access to ACMS.

Device Utility:
See Device Definition Utility.

Dictionary:
The VAX Common Data Dictionary. In the most general sense: an overall hierarchical storage facility that includes dictionary directories, subdictionaries, and objects.

Dictionary Directory:
The structure for organizing data descriptions stored in the CDD. Dictionary directories are similar in function to VAX/VMS directories. They "own" other dictionary directories or dictionary objects.

Dictionary Management Utility (DMU):
The Common Data Dictionary (CDD) management utility that lets you create and maintain the CDD directory hierarchy and its associated access control and history lists.

Dictionary Object:
Data definitions stored in the Common Data Dictionary. Examples of objects include:

- VAX DATATRIEVE domains, plots, procedures, and tables
- VAX DBMS schema, subschema, and storage schema definitions
- VAX CDD record definitions
- VAX TDMS requests and forms
- VAX Rdb/VMS relation, view, and field definitions

See also Common Data Dictionary.

Dictionary Verify/Fix Utility (CDDV):
A Common Data Dictionary utility that detects damaged dictionary files and repairs them. CDDV also compresses the data in dictionary files for more efficient use of storage.

Directory:
A list of files for a particular user.

See also default directory and slide directory.

Directory Hierarchy:
The structure of CDD directories. The hierarchy of dictionary directories, subdictionaries, and objects is similar to the hierarchy of relationships in a family tree. Each dictionary directory in the CDD tree may become a parent by owning other dictionary directories or dictionary objects. Dictionary objects are the terminal points of the hierarchy; they cannot be parents.

See also child, descendant, and parent.

Display Phase:
In DECgraph, the point at which you view a chart on the full screen. You use the DISPLAY icon or the PF3 key to start the Display phase.

Display Area:
The area of the screen, to the right of the menu areas, which DECslide uses to display its logo, longer help text, slide directories, and broadcast messages.

DML:
See data manipulation language.

DMU:
See Dictionary Management Utility.

Domain:
A DATATRIEVE data structure that associates a name with the relationship between a file and a record definition. Using the domain name gives access to information in the data file as interpreted by the record definition. For example, the domain PERSONNEL associates the file PERSON.DAT and the record definition PERSONNEL_REC.

Drop Shadow:
In bar charts, a contrasting portion of a bar that creates a three-dimensional effect. You use drop shadows to emphasize bars.

DYNAMIC Allocation:
In DBMS, an option that tells the Database Control System (DBCS) not to allocate space for an occurrence of a data item type until a run unit attempts to perform a DML operation on that item occurrence. DYNAMIC allocation also tells the DBCS to perform data compression on the item occurrence in the database. DYNAMIC allocation is declared in the storage schema.

Edit Plane:
In DECslide, the surface on which you work with objects to do your design work. An object in the edit plane appears in white.

See also finished plane.

Edit String:
A character or group of characters that directs DATATRIEVE to format a field in a specified way.

Elementary Field Description Statement:
A statement in a CDDL or DATATRIEVE record definition. An elementary field description defines a field that is not subdivided into subordinate fields.

Elementary Field:
A record segment containing one item of information. It might contain a department number, a last name, or any other information you want to define as a single item.

See also field.

Entry Point:
The point in a program where control is passed to a subroutine. The entry point used to pass control to DECslide is SLIDE$COMMAND.

See also callable interface.

Exchange Step:
One of three kinds of steps that define the work of a VAX ACMS multiple-step task. An exchange step handles input and output between the task and the terminal user.

See also processing step and block step.

Execution Controller:
See application execution controller.

Explicit Mapping:
The TDMS request instructions (INPUT TO, OUTPUT TO, and RETURN TO) that you specify to map data between form and record fields.

Export File:
A DECslide file that contains the graphics commands for displaying a slide without entering DECslide. You can type this file or mail it to another graphics terminal user.

External File:
A file opened in a main procedure that is accessed from an external subroutine.

External Subroutine:
A procedure that can be compiled separately from a main procedure.

Field:

- A segment of a data record.
- In TDMS, a single item on a form.
- In Rdb/VMS, a single division within a record where a data item is stored. You define a field's name and datatype, along with other characteristics, using the Rdb/VMS data definition language.
- In DECgraph, a single item or space on the Keyboard Data Entry Screen. You provide the information for the fields on this screen, such as a Y value, or accept a default value, such as the ONES value in the X units field.

See also elementary field and group field.

Field Attribute:
A condition or characteristic that defines the characteristics of the fields in a record. General field attributes provide unambiguous field descriptions recognized by every facility using the CDD. Facility-specific field attributes are supported by some languages or language processors, but not by others.

See also form field attribute.

Field Constant:
See form field constant.

Field Description Statement:
The statement that defines field characteristics in CDDL source files. The four types of field description statements are ELEMENTARY, STRUCTURE, COPY, and VARIANT.

Field Identifiers:
The characters specified in a TDMS form definition that determine the location, length, and picture-type of a field. For example, a C in the form definition of a field indicates that only an alphanumeric character (A-Z, a-z, space) can be entered in that field. The group of field identifiers that make up a field are the field picture.

See also field picture.

Field Name:
On DECgraph's Keyboard Data Entry Screen, a name that identifies a field, such as X data type, X units, Y units, Main Title, DTR, and so on.

In VAX DATATRIEVE use, the name given to a particular field in a record.

Field Picture:
A group of one or more field identifiers in a TDMS form definition that determines the location, length, and picture-type of a field. For example, 99999 is a field picture that indicates that up to five numeric characters can be entered in that field.

See also picture-type.

Field Validator:
In TDMS, a special field attribute that requires the terminal operator's input to be within a specified range, match an item from a specified list, or meet one or more other conditions. The form definer assigns field validators during the Assign phase.

Field Tree:
A hierarchical model of the fields in a DATATRIEVE record, based on the record definition stored in the Common Data Dictionary.

File:
A collection of related records.

File Name:
The name you choose to identify a file. The file name can have up to nine characters selected from the letters A through Z and the numbers 0 though 9. When you name files, you can give them any names that are meaningful to you.

File Specification:
A unique name used to identify a file. A full file specification identifies the node, device, directory name, file name, type, and version number under which a file is stored.

In DECslide, when you name a slide, you should not specify a file type or version number in answer to a file option prompt or in the SLIDE command string.

File Submenu:
In DECslide, an iconic menu with options for saving, deleting, restoring, and copying slides, for creating export files, and for viewing slide directories. The FILE icon in the main menu takes you to this submenu.

File Type:
The part of a file specification that describes the nature or class of file. The file type follows a period after the file name and consists of from one to three characters. For example, all DECgraph file types begin with .GR, such as .GRD for the data file and .GRL for the load file.

Filled Line Chart:
A line chart with the areas beneath the lines filled in with different colors or patterns. Filled line charts show volumes, such as sales volumes or a percentage of the whole market. Use the FILL icon to change a line chart to a filled line chart.

Finished Plane:
In DECslide, the surface on which you place the finished objects of a slide. Objects in the finished plane appear in black.

See also edit plane.

FIXED Member:
In DBMS, a record occurrence that, upon becoming a member of a set occurrence of a set type, must remain a member of that set until the record occurrence is erased from the database. Fixed set membership is specified in a schema DDL entry.

See also MANDATORY member and OPTIONAL member.

Foreign Key:
In Rdb/VMS, a key that does not uniquely identify records in its own relation, but is used as a link to matching fields in other relations. For example, DEPARTMENT_CODE is included in the JOB_HISTORY relation in order to link it to the DEPARTMENTS relation.

Form:
A preselected terminal screen image used to display and collect information.

Form Definition:
A description of a screen form, created in the TDMS Form Definition Utility (FDU) and stored in the CDD. A form definition can contain information that identifies:

- The screen image of the form, including the location of background text, fields, and video highlighting
- The length or size and data type of each field
- A set of attributes of each field on the form

Form Definition Utility (FDU):
The TDMS utility used to process (create, modify, replace, or copy) form definitions and to store them in the CDD. You also use this utility to provide a listing of form definitions used in an application.

Form Field Attribute:
A condition or characteristic, specified in a TDMS form definition, that applies to a field in a form. For example, a field attribute can require that an operator enter data into a field. Certain field attributes (for example, video characteristics) assigned in the form definition can be overridden by a request. The form definer assigns field attributes or accepts default attributes during the Assign phase.

Form Field Constant:
A character or embedded space that is displayed in a field on a TDMS form at run time. For example, you can use a hyphen as a field constant in a field that represents a telephone number. The form definer assigns field constants during the Layout phase.

Form Field Validator:
A special TDMS field attribute that requires the terminal operator's input to be within a specified range, match an item from a specified list, or meet one or more other conditions. The form definer assigns field validators during the Assign phase.

Forms Management System (FMS):
A DIGITAL software facility that lets you create forms and use them to display and collect information. You can associate a DATATRIEVE domain with an FMS form.

Free Space:
In DBMS, the sections of the database page that are not used.

See also database pages.

Frequency Distribution:
The relationship showing the number of times something occurs for a range of possibilities. Frequency is a limited number of distinct values, (such as the number of employees) and distribution is a range of continuous variables (such as the ranges of salary levels at a particular site). A histogram is the only type of chart that illustrates frequency distribution.

See also histogram.

Full Path Name:
A unique designation that identifies a dictionary directory, subdictionary, or object in the CDD hierarchy. The full path name is a concatenation of the given names of directories and objects, beginning with CDD$TOP, ending with the given name of the object or directory you want to specify, and including the given names of the intermediate subdictionaries and directories. The names of the directories and objects are separated by periods.

See also path name, relative path name, and given name.

Given Name:
The designation assigned to a dictionary directory, subdictionary, or object in the CDD. A given name contains up to 31 characters from the set A-Z, 0-9, _, and $. The first character must be a letter from A-Z, and the last character cannot be _ or $. The root directory, having only descendants but no ancestor, has been assigned the given name CDD$TOP. The given names of all other directories and objects are assigned by the user creating them. The given name of a dictionary directory, subdictionary or object is separated from the name of its parent by a period.

Global Aggregate:
In Rdb/VMS, an expression that uses field values from one relation to group records from another. A statistical expression is then used to calculate a value for the group. For example, you can group salary records in a SALARY_HISTORY relation according to the DEPARTMENT_CODE field in the DEPARTMENT relation. Then you can use the AVERAGE function to find the avera ge salary for each department.

GOLD Key:
The PF1 key on the numeric keypad that you use with DECslide in combination with some of the other keypad and arrow keys.

Graph Description File:
A DECgraph file that identifies the design choices you make for your chart, such as palette, type of chart, range, patterns, and so on. Graph description files have the file type .GRG.

Graphics Output File:
A DECgraph file that contains all the information necessary to reproduce a chart after you have finished a DECgraph session. You can use a graphics output file to mail a chart to another user or to display it on your own terminal without entering DECgraph. Graphics output files have the file type .GRO.

Graphics Slide:
Any DECslide slide composed of figures or of figures and text combined.

See also word slide.

Grid:
A set of fixed, regularly spaced vertical and horizontal lines.

In DECgraph, the grid lines can be vertical, horizontal, or both vertical and horizontal. Using a grid in DECgraph can help improve the understanding of data. You use the GRID options icon to choose a grid type.

Using a grid in DECslide can help with the alignment of your design work. You request the grid with the GRID ON key.

Group Data Item:
In DBMS, a named entity that contains one or more data item types.

Group Field:
A record segment containing one or more elementary fields. A group field may also contain other group fields. In the DATATRIEVE record definition PERSONNEL_REC, the group field NAME contains two elementary fields, LAST_NAME and FIRST_NAME. In TDMS request instructions, you can include the group field name to make a record field reference unique within the request.

A group field in DATATRIEVE is equivalent to a group data item in DBMS and a STRUCTURE field description in CDDL.

See also field, group data item, and STRUCTURE field description.

Group Record Array:
In TDMS, a record array whose elements contain other fields. Each of these fields has the same characteristics (length, data type, and so on), and each field is referenced in a request by the same name, but with a unique subscript. In request instructions, you can include the group array field name to make each field name unique.

Group Workspace:
A workspace that is primarily for holding information needed by many tasks in an ACMS task group. A group workspace is made available when the first task in a group that uses that workspace is selected by a terminal user. Once allocated, a group workspace remains available to all tasks in the group until the application stops.

See also workspace.

Hashing:
In DBMS, the conversion of a data item value (for example, a key value) into a fixed-length numeric value using a special algorithm. Hashed key values are used as physical pointers to database record occurrences.

HELP Key:
The PF2 key on the numeric keypad that gives you three levels of help for DECslide's icons, one level of help for the word list menus and prompts, and two separate keypad diagrams.

Hierarchical:
A type of database that organizes the relationships between record types as a tree structure. Related records are stored on the same branch of the tree to make data retrieval efficient.

Highlight:
An area of contrasting color that indicates your current position in a menu or screen.

Histogram:
A bar chart showing frequency distribution. Each bar is proportional in width to the range of values in a particular category (such as the salary levels at Plant X) and is proportional in height to the number of occurrences within each category (such as the number of employees). There are no spaces between bars, and the bars may or may not have equal widths, depending on the data.

History List:
An optional audit trail maintained by the CDD to monitor the processing and use of dictionary directories, subdictionaries, or objects.

See also audit trail.

Horizontal Label:
The name that identifies or describes the X values along the bottom of a DECgraph chart.

Icon:
A pictorial symbol that DECslide and DECgraph use to represent choices in menus.

Iconic Menu:
A menu that uses icons to present its choices. DECslide's main menu and file and print submenus are iconic menus.

Identification File:
A DECgraph file containing the text that identifies the data you enter to create a chart, such as title, subtitle, horizontal label, and so on. An identification file and data file make up a data set. Identification files have the file type _.GRI.

Image:
A file consisting of procedures and data that have been bound together by the linker. There are three types of VMS images: executable, shareable, and system. When not otherwise stated,image refers to an executable image.

Implicit Mapping:
An instruction (OUTPUT %ALL, INPUT %ALL, or RETURN %ALL) in a TDMS request that lets you map data between all identically named form and record fields without specifying the individual fields. If the Request Definition Utility finds an error in an implicit mapping, it does not include that mapping when it stores the request in the CDD. At run time, TDMS performs only the correct mappings.

See also explicit mapping.

Index:
A structure within a file or database that allows you to locate particular records based on field values.

Index Key:
A field of a record in an indexed file or database that determines the order of search and retrieval.

- An RMS indexed file has one primary key and optionally one or more alternative keys.
- In DATATRIEVE, you declare an index key in the DEFINE FILE command, by naming a field from a record definition.
- In Rdb/VMS, you can use any field or combination of fields from a record as an index key. You can also define more than one key for a given relation.

INDEX Mode Set:
In DBMS, a sorted set in which a hierachical index data structure is used to speed access to a specified record occurrence. INDEX mode is specified in a storage schema DDL entry.

See also index node.

Index Node:
A DBMS data structure within a hierachical index that provides additional speed of access.

See also index mode set.

Indexed File:
A file that contains records and a primary key index (and optionally one or more alternate key indices) used to process the records sequentially by index or randomly by index.

Indexed Form Array:
A list of elements on a TDMS form, all of which have the same name and the same characteristics (length, data type, and so on). The form definer specifies how many elements the array contains.

Initialization Procedure:
In ACMS, a procedure that runs when a procedure server process starts and that usually opens files or readies a database for the server process.

Input Parameter:
In DECslide, the part of the SLIDE command string used to specify the input slide name for the print, display, restore, delete, and copy options.

Insert Mode:
The default text entry mode in DECslide that lets you insert new text.

See also replace mode.

INSERTION Class:
In DBMS, an attribute of member record types that describes how and when member record occurrences are added to set occurrencess.

See also AUTOMATIC member and MANUAL member.

Integrity:
Measures taken to ensure the correctness of information in an Rdb/VMS database. The three general functions of integrity are integrity constraints, concurrency control, and recovery.

- Integrity constraints ensure that database information remains correct when users attempt to modify it incorrectly.
- Concurrency control allows only one user at a time to update a file while allowing many users simultaneous access to the database.
- Recovery restores a database to a known state prior to a system failure.

Interactive Mode:
The default mode for running DECslide and DECgraph.

In DECslide, interactive mode lets you use the menus and design options to create and modify slides. For an interactive session, you can use the command string to specify a new default palette and input and output parameters.

In DECgraph, interactive mode lets you use the menus to enter or change information for your graphs. You can specify a particular data set or graph description file as defaults for that session as well as supply other information.

See also noninteractive mode.

Interactive Processing:
A mode of computer operation in which the commands and data that control the actions of the computer are entered by a person at a terminal rather than by a programmed script.

Interactive DBQ:
In DBMS, a data manipulation interface to the Database Control System that allows low-volume, interactive access to a database. You can use interactive DBQ as a tool to test and debug program logic. When used on a VT100 terminal, interactive DBQ uses a split screen to show your current position in a subschema after each DML statement is executed.

See also database query and Callable DBQ.

Interpretive Call Interface:
See Callable RDO.

Isolate:
To pull out a segment of a DECgraph pie chart for emphasis. Use the ISOLATE options icon to choose whether or not to isolate segments.

Join Operation:
A procedure that selects a record from one relation, associates it with a record from another relation, and presents them as though they were part of a single record.

Journal File:
In DBMS and Rdb/VMS, a file that contains all records modified by a run unit or transaction. The journal file allows reconstruction of the database in the event of corruption due to system or program failures.

Journaling:
The process of recording information about operations on a recoverable resource, such as a database. The type of information recorded depends on the type of journal being created.

See also after-image journal and before-image Journal.

Junction Record:
In DBMS, a record that relates two records to each other. You can use a junction record to define a recursive or many-to-many relationship between two records.

Keeplist:
In DBMS, a list of record identifiers used to recall their associated records. Identifiers are placed on and removed from keeplists at the direction of a DML operation.

Key:
In Rdb/VMS, a field in a record that you use to define an index. Using index keys, Rdb/VMS can locate records in the relation directly, without searching sequentially. Defining index keys increases the speed of some database operations.

See also index key, candidate key, and foreign key.

Key Value:
In DBMS, the values supplied in a DML operation to identify a specific record for access.

Keyboard Data Entry Screen:
One method of data entry in DECgraph. A form on your terminal screen with fields for the values, titles, and labels that make up the information for your chart. You can enter or change information using the Keyboard Data Entry Screen. Use the KEYBOARD icon to use the Keyboard Data Entry Screen.

Keyword:
A word reserved for use in certain specified syntax formats, usually in a command or a statement.

In DECgraph load files, keywords tell DECgraph what type of information (text or data) follows. For example, the keyword TITLE tells DECgraph to expect text.

Legend:
A small box outlined in a DECgraph chart that contains the legends you entered for the Y1 through Y6 legend fields in the Keyboard Data Entry Screen. The legend fields identify the Y values; in the Annual Reports data, for example, the Y1 legend Net identifies one particular value that will be plotted for several years.

See also Y1-Y6 legends.

Line Chart:
A chart in which the conjunction of the X and Y values is signified by points plotted in relation to the X and Y axes. The points are connected to each other to form a continuous line.

See also filled line chart.

Line Index:
In DBMS, a dynamic section of a database page that acts as a directory to data on the database page.

See also database pages.

Line Type:
A line pattern, such as a dotted, dashed, unbroken, or double-thick line, used with DECgraph in line charts, trend lines, and so on. DECgraph provides nine line types for designing charts. You use the LINE TYPE icon to choose a line type.

Linear Scaling:
A method of arranging the intervals between values on an axis. In DECgraph, linear scaling shows the minimum and maximum values on an axis with 6 to 11 equal intervals in between. DECgraph automatically numbers these intervals with appropriate numbers.

Literal:
A value expression representing a constant. A literal is either a character string, enclosed in quotation marks, or a number.

Load File:
A DECgraph file that you can use as one method for data entry and that contains keywords identifying the type of data (TITLE, X_UNITS, and so on) and the data values. The load file option lets you write programs in a simple format to generate large files of data.

Locking:
In DBMS and Rdb/VMS, the facility that allows concurrent access to a database as a whole, without allowing concurrent access to the same record. VAX DBMS allows locks on individual records, entire realms, or both.

Logarithmic Scaling:
A method of arranging the intervals between values on an axis. In DECgraph, shows equally spaced intervals on an axis for each magnitude. For example, the space between 1 and 10 is equal to the space between 10 and 100.

MANDATORY Member:
In DBMS, a record occurrence that, upon becoming a member of a set occurrence of a particular set type, must remain a member of that set type until the record is erased from the database. MANDATORY set membership is specified in a schema entry. It can be moved from one set occurrence to another.

See also FIXED member and OPTIONAL member.

MANUAL Member:
In DBMS, a record occurrence that becomes a member of a specific set occurrence by direction of an application program. MANUAL set membership is specified in a schema entry.

See also AUTOMATIC member.

Main Menu:
In DECgraph and DECslide, the group of icons indicating your major choices. DECgraph's main menu gives you choices for creating and designing charts and contains the STOP, HELP, DATA, DESIGN, and DISPLAY icons. DECslide's main menu contains the STOP, HELP, MESSAGE, FILE, DESIGN, PRINT, and ORGANIZE icons.

Mapping:
The description of the exchange of data between a TDMS form and a program record and/or a request.

Marker Path:
The set of DECgraph markers for a given Y value. A marker appears at the appropriate Y value point for each X value point.

See also scatter chart.

MDB:
See menu database.

Member Record:
In DBMS, a record type, other than an owner record type, included in a set type. There may be one or more member record types in a set type, and zero or more record occurrences in each member record type. A record occurrence in a member record type is not directly accessible by the DBCS, unless it is also the owner of another set. It must be accessed through its owner record occurrence or the SYSTEM record.

See also owner record type and nonsingular set type.

Menu:
A list of tasks, from which a user selects one for processing. A menu can also direct users to other menus.

In ACMS, you define the list of items on a menu and other menu characteristics using the Application Definition Utility.

In all of its menus, DECgraph uses icons instead of words to list menu choices. DECslide uses both iconic and word list menus to present its options.

See also icon, main menu, submenu.

Menu Area:
The area at the top left of your screen that DECslide uses to present its main menu icons.

Menu Database:
A run-time database containing information derived from menu definitions. ACMS uses the information in the menu database for displaying menus. A menu database is created by building menu definitions with the Application Definition Utility (ADU).

Message File:
A file that contains a table of message symbols and their associated text.

Message Flag:
A blinking signal (*M*) that DECslide displays at the right of the message line to indicate when a message has arrived.

See also broadcast message.

Message Line:
The space below the work area that DECslide uses to display its word list menus, instructions, prompts, and status and help messages.

Multiple-step Task:
An ACMS task defined in terms of a block step that contains one or more exchange and processing steps.

See also block step, exchange step, processing step, and step action.

Nested Conditional Instruction:
A conditional instruction contained within an outer conditional instruction. TDMS executes the inner conditional instruction only if it executes the associated outer conditional instruction.

Node:
In the CDD message files, the word node is sometimes used as a generic name for dictionary directories, subdictionaries, and objects.

Noninteractive Mode:
One way in which you can run DECslide or DECgraph. You use the /NOINTERACTIVE qualifier in the command line to specify this mode.

Using noninteractive mode with DECslide lets you display and print slides in different palettes and create export files without entering DECslide.

Using noninteractive mode with DECgraph lets you display and print graphs and create graphics output files without entering DECgraph.

See also interactive mode.

Nonsingular Set Type:
In DBMS, a set type owned by a user-defined record type, not by the SYSTEM record.

See also SYSTEM-owned set, member record type, and owner record type.

Normalization (of a database):
The process that physically separates related concepts in the database into separate relations. Normalization ensures that separate concepts are kept physically separate in the database. In this way, a data item is stored only once. You only need to perform one update operation to change it. When you need to bring data together from different relations or when you want an employee's job history, the database allows you to create temporary relationships by joining relations together.

Novalidate Mode:
The mode in the TDMS Request Definition Utility that lets you create and store a request without checking for correct mappings and references. You create a request in Novalidate mode by using the SET NOVALIDATE command. Validate mode is the default.

See also validation.

NUMERIC Data Type:
In DECgraph, a field value indicating that the X values are to be considered as numbers rather than text.

See also TEXT data type, X data type.

Object:
See dictionary object.

Object Menu:
A DECslide word list menu with options that give you ready-made shapes or that let you create your own shapes or enter text.

Operator Command:
See ACMS Operator Command.

OPTIONAL Member:
In DBMS, a record occurrence that can be removed from all set occurrences. You can change its set occurrence membership without deleting it from the database. OPTIONAL set membership is specified in a schema entry.

See also FIXED member and MANDATORY member.

Output Parameter:
In DECslide, the part of the SLIDE command string used to specify the name of the new file for a copied slide, a saved slide, or an export file.

Overlay:
To restore a saved DECslide slide and place it on the slide currently in the work area, combining the two slides into one.

Owner Record Type:
In DBMS, the record types that access entry points to the set occurrences. There can be only one record type as the owner for each set type and one owner record occurrence for each set occurrence.

See also member record type, nonsingular set type, and SYSTEM-owned set.

Page Header:
In DBMS, a fixed-length section at the beginning of the database page that contains page and storage area information.

See also database pages.

Paint Menu:
A DECslide word list menu with options for palette, color, and object selection that you use to add color to a slide. This menu also includes an option for changing the default background color and another option for checking the color st atus of objects in the slide.

Palette:
One of 12 sets of 10 color combinations used with DECslide slides and DECgraph charts. The default palette is Palette 1.

In DECslide, each palette has four solid colors and six screened colors, which are percentages of the solid colors and which you use only to paint solid filled objects.

Parameter:
The object of a command. A parameter can be a file specification, a keyword option, or a symbol value.

Parent:
A way of describing a dictionary directory or subdictionary in the CDD that immediately precedes a directory, subdictionary, or object in the CDD hierarchy. The preceding directory or subdictionary is called the parent. The parent is said to own its children. A parent may have many children, but each dictionary directory, subdictionary, and object in the CDD may have only one parent. For example, CDD$TOP is the parent of CDD$TOP.MANUFACTURING. CDD$TOP owns CDD$TOP.MANUFACTURING.

See also child and descendant.

Partial Path Name:
See Relative Path Name.

Path Name:
A unique designation that identifies a dictionary directory, subdictionary, or object in the CDD hierarchy. The full path name joins the given names of directories and objects, beginning with CDD$TOP, ending with the given name of the object or directory you want to specify, and including the given names of the intermediate subdictionaries and directories. The names of the directories and objects are separated by periods.

A path name may be:

- Full, beginning with CDD$TOP. For example, the full dictionary path name for a DATATRIEVE domain might look like this: CDD$TOP.DEPT32.EMPLOYEE.
- Relative, beginning with the name of a child of your default dictionary directory. For example, if your default directory is CDD$TOP.DEPT32, EMPLOYEE would be a relative path name referring to CDD$TOP.DEPT32.EMPLOYEE.
- Logical, using a name you have defined for a full or relative path name. For example, you might use the following DCL command:

  $ DEFINE EMP CDD$TOP.DEPT32.EMPLOYEE

  Then, within the current process, EMP would be equivalent to CDD$TOP.DEPT32.EMPLOYEE.

See also given name and relative path name.

Pie Chart:
A circular chart with wedge-shaped segments indicating the X values. The size of the wedges in relation to the whole indicates the Y values as percentages. In DECgraph, segments can be pulled out or isolated to emphasize a particular portion of the data.

Pixel:
The smallest displayable unit, or picture element, on a video display screen.

In DECslide, the work area measures 767 by 454 pixels. The PIXEL TOGGLE key sets the status for cursor movement from COARSE (50 pixels) to MEDIUM (10 pixels) to FINE (1 pixel).

DECgraph gives you the icons FINE, MEDIUM, and COARSE for controlling cursor movement during the assignment phase.

PLACEMENT Mode:
In DBMS, a storage method by which the DBCS determines the database key values associated with record occurrences based on user-specified set options. PLACEMENT mode is declared in the storage schema.

See also SCATTERED set option and CLUSTERED VIA set option.

Pointer:
In DBMS, a place marker that identifies a record's address in a storage area.

See also database key.

Polygon:
In DECslide, a multisided, closed shape that you can draw using the POLYGON option from the object menu.

Precompiler:
A utility that reads Rdb/VMS statements in a high-level language program and translates those statements into calls to low-level Rdb/VMS routines.

Primary Key:
In an indexed file, the index key whose value determines the order of records. You cannot modify or erase the value in a primary key field of a DATATRIEVE record.

Print List:
One or more value expressions (including the names of elementary and group fields) whose values you want DATATRIEVE or Rdb/VMS to display. A DATATRIEVE print list can also include optional formatting specifications.

Print Submenu:
A DECslide iconic menu with two options for printing slides: single or double size. The PRINT icon in the main menu takes you to this submenu.

Privilege:
A characteristic of a user that controls that user's ability to access a file or other resource for a certain purpose. Thirteen privileges have been defined to control access to the CDD. Four of these privileges are specific to VAX DATATRIEVE; the remaining nine are VAX CDD access privileges.

See also access control list.

Procedure:

- A general purpose routine, entered by means of a call instruction, that uses an argument list passed by a calling program and uses only local variables for data storage. A procedure is entered from and returns control to the calling program.
- A fixed sequence of DATATRIEVE commands, statements, clauses, or arguments that you create, name, and store in the Common Data Dictionary.
- A series of Rdb/VMS RDO statements stored in a VMS file. These can be executed with the execute (@) directive.

See also step procedure, initialization procedure, and termination procedure.

Procedure Object Library:
A collection of object modules for procedures. An object module is the binary output of a language processor such as the assembler or compiler. The linker accepts object modules as input.

Procedure Server:
One of two types of servers that handle processing work for ACMS tasks.

See also server, DCL server, and procedure server image.

Procedure Server Image:
The image that is loaded into a procedure server process when the process is started by the ACMS execution controller. The procedure server image is created when all the procedures handled by the server are linked together with the procedure server transfer module for that server.

See also DCL server image and procedure server transfer module.

Procedure Server Process:
See server process.

Procedure Server Transfer Module:
The object module created for a procedure server as a result of building an ACMS task group definition. When a task group is built, the Application Definition Utility produces a procedure server transfer module for each server defined in the task group. The procedure server transfer module is linked together with all the procedures handled by the server to produce the procedure server image.

Process:
The entity scheduled by the VMS system software that provides the context in which an image runs. A process consists of an address space and both hardware and software context.

Process Context:
See server context.

Processing Step:
One of three kinds of steps that define the work of a task defined with ACMS. The work of a processing step is handled by a server and can consist of computations, data modification, file and database access.

See also block step and exchange step.

Program Interface:
A set of callable CDD routines. You can use the CDD program interface to build software products that use CDD functions.

Program Request Key (PRK):
In TDMS, a key or combination of keys from the VT100 keyboard or keypad that you can define in a request to allow the terminal operator to communicate with the application program at run time.

A PRK can be one of the following:

- The keyword KEYPAD followed by one key (0-9, hyphen, period, or comma) from the VT100 keypad
- The keyword GOLD followed by one printable key from the main keyboard (including the space bar, but not the tab key)

Project:
See reduction operation.

Prompt:
A question that DECslide asks when you are to name a slide or verify a file option or some other operation used with DECslide, such as when you use the ERASE key to erase the work area.

Prompting Expression:
An expression that directs DATATRIEVE to ask the user to supply a value when a statement is executed.

Qualifier:
A portion of a command string that modifies a command verb or command parameter. A qualifier follows the command verb or parameter to which it applies and has the following format: /qualifier[=option].

Query Header:
A substitute column header that replaces the field name when DATATRIEVE displays values from a field. For example, "STATUS" appears at the top of the column listing values from the field EMPLOYEE_STATUS.

Query Name:
A synonym you give to a DATATRIEVE field name in order to make input easier to type and remember. For example, you can type L_NAME instead of LAST_NAME when using the PERSONNEL domain.

Quiet Point:
In DBMS, a time when a run unit is not accessing any database areas. Quiet points occur between transactions.

See also transaction.

RDO:
See Relational Database Operator.

RDU:
See Request Definition Utility.

RDU Commands:
The commands you issue to operate the TDMS Request Definition Utility, including commands to process (create, modify, copy, delete, and so on) a request or a request library definition.

Range:
In DECgraph, the extent of the values on the X and/or Y axis from the minimum to the maximum value.

Read Access:
File protection that you set to let a user read a file from your VMS directory. This protection must be set, for example, when you use DECslide's file and print options to specify another user's slide.

Realm:
In DBMS, one or more areas grouped to allow subschema access. Realms are specified in a subschema entry.

See also area.

Record:

- A body of related information that is the basic unit for storing data.
- In Rdb/VMS, a collection of related fields stored as a unit in a relation. For example, one employee record may include an employee's name, address, department, salary, and starting date.
- In DECgraph, a set consisting of an X value and all the Y values for that X (Y1 through Y6).

See also field, record occurrence, and record type.

Record Definition:
The description of a record's structure that includes the name, data type, and length of each field. CDDL, DATATRIEVE, and DBMS all store record definitions in the CDD. ACMS, TDMS, COBOL, BASIC, DIBOL, and PL/I access record definitions stored in the CDD.

Record Management System (RMS):
A set of VMS operating system procedures that programs can call to process files and records within files. VAX RMS lets programs issue GET and PUT requests at the record level (record I/O) as well as read and write blocks (block I/O). RMS is an integral part of the VMS system software and is used by high-level languages, such as VAX COBOL and BASIC, to implement their input and output statements.

DATATRIEVE uses VAX RMS to create, define, store, and maintain files and records within files.

Record Number:
On the DECgraph Keyboard Data Entry Screen, the number that appears below each X value indicating the specific record (01, 02, 03 and so on through 9999). You can have any number of records, but those after 9999 will have **** instead of a number.

Record Occurrence:
A group of related data item occurrences that is the basic unit for accessing data in a database. In DBMS, the definition of a record occurrence is the record type.

See also data item occurrence and record type.

Record Selection Expression (RSE):
A phrase that defines specific conditions that individual records must meet before Rdb/VMS or DATATRIEVE includes them in a record stream. The RSE allows you to determine the subset of records to be retrieved from a set of domains or a database.

Record Stream:
A group of records defined by a record selection expression. This group lasts only during the execution of the statement that forms it.

In Rdb/VMS, a record stream is formed by either a FOR or a START_STREAM statement. Streams are used in an application program or RDO to retrieve one record at a time for manipulation.

See also record selection expression (RSE).

Record Type:
A group of related data item types that defines a record occurrence. In DBMS, record types are specified by a schema entry and modified in the subschema.

See also data item type and record occurrence.

Recovery:
In DBMS and Rdb/VMS, the process of restoring a database to a known condition after a system or program failure.

In ACMS, you can define recovery as a characteristic for a multiple-step task that uses VAX DBMS.

See also after-image journal, before-image journal, journal file, journaling, and transaction.

Recovery Unit:
A recovery unit groups operations on recoverable resources such as databases. All operations on recoverable resources in a recovery unit are done indivisibly; either all the operations occur or none of the operations occur.

See also transaction.

Recovery-unit Journal:
See before-image journal

Reduction Operation:
In Rdb/VMS and DATATRIEVE, an operation that finds the unique values for a field or group of fields and eliminates repeated records. Reduction is sometimes called the project operation. Use the REDUCED TO clause to perform the operation.

Reflexive Join:
An operation that joins a relation to itself.

Relation:
A method of presenting related data that consists of a set of rows and columns. The columns have names and divide each row into a set of fields. For a single field in a row, there is only one data item. In VAX Rdb/VMS, columns are referred to as fields, and rows are called records. A relation is sometimes called a table.

Relational Database:
A relational database represents data as a set of independent relations, or relations. Within a relation, data is organized in columns and rows, with at most one data item occupying each intersection. Relationships between relations depend on values within the relations.

Relational Database Operator (RDO):
A single interactive utility for maintaining the database, creating and modifying definitions of database elements, and storing and manipulating data.

See also Callable RDO.

Relational Operators:
Symbols, keywords, or phrases you can use to compare values. For example, the DATATRIEVE statement, FIND PERSONNEL WITH SALARY > 10000, contains the relational operator > (greater than).

Relative Path Name:
The shortened form of a dictionary path name. It includes only the parts of the path name that follow the default CDD directory name. You can use either the full path name or the relative path name to refer to directories, subdictionaries, and objects in the CDD.

See also given name and path name.

Remote Server:
The part of DATATRIEVE that lets you access data on other computers. If you are using the computer VACKS1 and you type READY PERSONNEL AT VACKS2, DATATRIEVE logs on to an account on VACKS2. The Remote Server processes your statements at the remote computer VACKS2.

Replace Mode:
The text entry mode in DECslide that lets you replace existing text with new text. You use the INSERT/REPLACE key, a toggle key, to change from the default insert mode to replace mode.

See also insert mode.

Report Header:
The heading of a DATATRIEVE Report Writer report, consisting of these optional elements: a centered report-name and, at the top-right corner of the report, a date and a page number.

Report Specification:
A series of DATATRIEVE Report Writer statements that create a report and specify its format.

Report Writer:
A subsystem of DATATRIEVE that lets you create reports displaying data in an easy-to-read format.

Request:
A set of TDMS instructions, created in the Request Definition Utility and stored in the CDD, that describes an exchange of data between a program record and a form. A request includes references to one or more form and record definitions and instructions for mapping data between a form and a program record. A request is passed as a parameter in the TSS$REQUEST call.

ACMS tasks use requests to display forms on a terminal and gather information from a terminal user.

Request Call:
The call in a TDMS application program that executes a request.

Request Definition Utility (RDU):
The TDMS utility used to process (create, modify, replace, and so on) requests and request library definitions and to store them in the CDD. You also use this utility to build request library files, which are accessed by an application program at run time.

Request Instructions:
The statements in a TDMS request that describe the exchange of data between a program record and a form.

These statements can:

- Identify the form and record definitions between which data is to be transferred
- Provide instructions for transferring the data

The request instructions are executed when the TDMS application program issues a TSS$REQUEST call.

Request Library Definition:
A definition, stored in the CDD, that lists the names of related requests to use in a particular TDMS application. A request must be named in a request library definition before you can build a request library file. The program uses the request library file to access requests.

Request Library File (RLB):
A VAX/VMS file that contains TDMS requests and the form and record information necessary to execute these requests. When you use the Request Definition Utility to build a request library file, RDU reads the definitions in the CDD and puts information in the request library file so that the program can execute the requests. A request library file that contains a request named in a TDMS call must be opened before a program can use the request.

Request Library Instructions:
The statements in a TDMS request library definition that identify the requests used in a TDMS application. These instructions also give the name of the request library file where these requests and their associated form and record definitions are to be stored.

Restore:
In Rdb, an operation that rebuilds a database from a saved copy after a hardware or software failure.

In DECslide, to return a saved slide to the work area so you can view it, modify it, or reuse it by overlaying it on the current slide.

Restrict:
See select.

Restriction Clause:
A phrase in the DATATRIEVE record selection expression that allows you to specify the maximum number of records making up a record stream.

RETAINING:
In DBMS, an option on the DML COMMIT statement. The COMMIT RETAINING statement:

- Does not empty keeplists.
- Retains all currency indicators.
- Does not release realm locks.
- Releases all record locks.

RETENTION Class:
In DBMS, an attribute of member record types that describes when and how a member record occurrence can be removed from a set.

See also FIXED member, MANDATORY member, and OPTIONAL member.

RLB:
See Request Library File.

RMS:
See Record Management Services.

Rollback:

- In DBMS or Rdb/VMS, the process of using a before-image journal to restore a database to an earlier known state. This process negates updates to the database made by the transaction or recovery unit being rolled back.
- In ACMS, an Application Definition Utility keyword used when defining multiple-step tasks with recovery.

Rollforward:
In DBMS and Rdb/VMS, the process of using an after-image journal to restore a database to a known state. This process replaces updates to the database that were lost because a system or program failure required the installation of backup media.

See also recovery.

Root Dictionary Directory:
The directory at the top of the VAX CDD hierarchy. The root directory is named CDD$TOP. Every dictionary directory, subdictionary, and object in the CDD is a descendant of CDD$TOP.

Row:
In DECgraph, a set of numbers or words arranged horizontally rather than vertically. For example, on the Keyboard Data Entry Screen, the Y values for the Y1 legend appear in a row across the screen.

Row (of a table):
See record, relation.

Run Unit:
In DBMS, an execution of a single program that accesses a database.

Scaling:
A method of arranging the intervals between values on an axis. DECgraph provides three scaling choices: linear, logarithmic, and no scale.

Scatter Chart:
A chart that displays markers to indicate the data progression for Y values. A marker appears at the appropriate Y value poit for each X value point.

SCATTERED Set Option:
In DBMS, A record placement option in which records are evenly distributed throughout database pages, based upon data values in the record. SCATTERED mode is specified in a storage schema entry.

Schema:
In DBMS, the logical description of a database, including data definitions and data relationships. The schema is written using the schema data definition language (schema DDL).

Schema Data Definition Entry:
In DBMS, the entry of the Data Definition Language (DDL) used to define the logical structure of a database.

Scientific Notation:
A way of expressing very large or very small numbers as a constant multiplied by the appropriate power of 10. For example:

.000000009	.9E-8 (1 times 10 to the power of -8)
9000000.	.9E 7 (1 times 10 to the power of 7)

Screened Color:
One of six colors in a DECslide palette that is a percentage of two of the four solid colors in this palette. You use a screened color to paint solid filled objects only.

Scrolled Form Array:
A list of elements in a scrolled region on a TDMS form, all of which have the same name and the same length and data type. The form definer does not specify the number of elements in the scrolled region, and the request definer can map up to 32,767 elements of data.

Scrolled Region:
An area, specified on the TDMS form definition, that permits the terminal operator to move through many lines on a field and view or enter data, although only a few lines appear at one time on the screen.

Security:
Measures taken to protect the information stored in a database against unauthorized reading, writing, or deletion.

Select:
In Rdb/VMS and DATATRIEVE, an operation that discards records that do not satisfy a conditional expression. For example, if you want to display employees with salaries greater than $20,000, a selection operation prevents employees records with salaries less than or equal to $20,000 from appearing in the output. You perform this operation using the WITH clause of the record selection expression.

Selected Record:
In a DATATRIEVE collection, the one record marked by the SELECT statement and available for display or modification without specifying a record selection expression.

Selector Box:
In DECslide, a highlighted box used to frame a selection in the word list menus and in a slide directory listing in the display area.

Sequential File:
A file whose records appear in the order in which they were originally written. A sequential file does not have an index. In DATATRIEVE, you cannot delete records from a sequential file.

Server:
The ACMS component that handles processing work for a task. There are two types of servers: DCL servers and procedure servers. The implementation characteristics for a server are defined in a task group definition. The operational characteristics for a server are defined in an application definition.

See also DCL server and procedure server.

Server Command:
The string passed by an ACMS application execution controller to a server process at the start of a processing step. The string identifies what work the server is to perform.

Server Context:
In ACMS, information local to a server process, such as record locks and file pointers. Process context can be retained from one step to another in a block step but cannot be passed between servers or tasks.

Server Image:
A VMS image that the ACMS run-time system loads into a server process. There are two types of server images: DCL server image and procedure server image.

Server Process:
A VMS process created according to the characteristics defined for a server in an ACMS application and task group definition. Server processes are started and stopped as needed by application execution controllers.

Set:
A defined relationship among records in a DBMS database. A set contains an owner record, one or more member record types, and zero or more member record occurrences.

See also set occurrence and set type.

Set Occurrence:
In DBMS, an occurrence of a set type. A set occurrence consists of one record occurrence from an owner record type and one record occurrence from zero, one, or more different member record types.

Set Type:
In DBMS, a definition of a relationship that exists among record types in a database. A set type contains an owner record type, and one or more member record types.

See also set occurrence.

Simple Record Array:
See array.

Single-step Task:
An ACMS task that has only a single processing step. Single-step tasks can be defined in a task group or a separate task definition.

See also multiple-step task.

Singular Set:
See SYSTEM-owned Set.

Size Validators:
A field validator on a TDMS form definition that determines the field data type and sets a predefined range for numeric fields. At run time, size validators prevent the operator from entering data that is not within that range. The form definer assigns size validators in the Assign Phase.

Slide File:
The file that DECslide creates when you save a slide design.

Slide Directory:
In DECslide, an alphabetized list of saved slides. Each item in the list includes the slide name, the comment entered by the user, and the date and time when the slide was saved.

Software Event Logger (SWL):
The TDMS process that records ACMS and TDMS software events that occur during the running of an application program. In order to see the events logged by the SWL, you must use the Software Event Logger Utility Program.

See also Software Event Logger Utility Program.

Software Event Logger Utility Program (SWLUP):
The TDMS utility you use to list selected events that were logged by the Software Event Logger.

Solid Color:
One of four colors in a DECslide palette that you use to paint the objects, text, and background in a slide. By default, DECslide uses the first solid color for the background, the second for the objects, and the fourth for the text. The third is the color you can use by default to change any other colors in a slide.

Sort Key:
A field that forms the basis for sorting. For example, you can rearrange the records in DATATRIEVE's sample domain PERSONNEL according to seniority by typing PRINT PERSONNEL SORTED BY START_DATE.

Sorted Set:
See INDEX mode set.

Stacked Bar Chart:
A chart that portrays data on more than one item in each single bar. Each data item is distinguished by a different pattern or color.

Statement:
A string of characters that a user or program transmits to a software product to execute a function.

In DATATRIEVE, a statement is distinct from a command. A statement performs query, report, or data manipulation functions. Two or more statements can be joined, while a command must stand alone.

See also command.

STATIC Allocation:
In DBMS, the default allocation option of the RECORD statement of the Storage Schema entry. Use it to specify the amount of physical storage you want to dedicate to a particular data item type. You make the specification during the definition of the database, but the actual allocation does not occur until the creation of the database.

See also DYNAMIC allocation and storage schema.

Statistical Expression:
In DATATRIEVE and Rdb/VMS, an expression that takes values from multiple rows of a relation and combines them into a single result. Statistical expressions include AVERAGE, MAX, MIN , COUNT, and TOTAL.

Status Message:
A one-word message that DECslide displays in a box to the right of the message line to indicate the operation you have selected most recently. For example, the word OBJECT appears in the box when you press the OBJECT key to enter the object menu.

Step:
A part of an ACMS task definition that identifies one or more operations to be performed. Task definitions can have three kinds of steps: block steps, processing steps, and exchange steps. Each step contains clauses that describe the work to be done in that step and the action that follows the work.

See also block step, exchange step, processing step, step action, step work, single-step task, and multiple-step task.

Step Action:
The part of a step definition that tells ACMS what to do after completing the work for that step. These instructions can consist of a single unconditional action or a series of conditional actions based on the value of a field in a workspace.

Step Label:
A name assigned to a step in a multiple-step ACMS task.

Step Procedure:
A type of procedure called in a processing step of an ACMS task. Step procedures handle computations, data modification, and file and database access for processing steps that use procedure servers. Normally, step procedures do not handle input from or output to a terminal.

Step Work:
The part of an ACMS step definition that describes terminal interactions, processing, or both.

Storage Schema:
In DBMS, a description of the physical storage of data in a database. The storage schema is written using the storage schema data definition entry.

See also storage schema data definition entry.

Storage Schema Data Definition Entry:
In DBMS, the entry of the Data Definition Language (DDL) used to define the physical organization of a database.

String Descriptor:
A data structure that specifies the address, length, and data type of a string. String descriptors are passed as arguments to subroutines.

STRUCTURE Field Description Statement:
In CDDL, STRUCTURE field description statements define fields that are subdivided into one or more subordinate fields. The top-level field description statement for a record is ordinarily a STRUCTURE field description statement.

Subdictionary:
In the CDD hierarchy, subdictionaries function almost exactly as if they were dictionary directories, but they exist as physically separate dictionary files.

Subdirectory:
A list of files that is grouped one or more levels below the top-level or main VMS directory.

If you are using DECslide, you should run it from a separate subdirectory, especially when you are creating several slides for a presentation.

Submenu:
A group of menu options generated when you select a main menu option.

In DECslide, for example, you enter a submenu when you select the FILE or PRINT icons in the main menu.

In DECgraph, you can choose the DATA icon in the main menu to enter the Data Entry submenu, where you can choose the LOAD icon.

Subobject:
In DECslide, an object that is joined with other objects to form a compound object. You use the JOIN key to form a compound object and the GOLD-SEPARATE key to separate subobjects from a compound object.

Subschema:
In DBMS, a tailored, user-oriented view of a database. A view may be tailored to meet the needs of a particular programming language or to focus the kind of data a program can access to that specifically required to perform an end user task. The subschema can include everything in the corresponding schema or any part of the schema. The subschema is written using the subschema data definition entry.

See also subschema data definition entry.

Subschema Data Definition Entry:
In DBMS, the entry of the Data Definition Language (DDL) used to define user-oriented views of a database.

Subscript:
A positive integer that indicates the position of an element in a form or record array. For example, in a TDMS request instruction, to refer to the third element of an array LAST_NAME, you use the array field name and the number 3 (indicating the third element): LAST_NAME[3].

Substitution Directive:
An expression in a command or statement passed to DATATRIEVE from a calling program. The substitution directive is replaced by parameters given in the program.

Subtitle:
In DECgraph, a secondary line that appears beneath the main title of your chart. Use the subtitle to further clarify the main title information.

Summary Lines:
Information you can display in a DATATRIEVE report with the AT TOP and AT BOTTOM statements.

SWL:
See Software Event Logger.

SWLUP:
See Software Event Logger Utility Program.

Synchronous Call:
A call to a TDMS subroutine that performs the entire requested action before your program can continue running. Thus, your program continues only after the completion of the called subroutine. Most calls are synchronous calls.

See also asynchronous calls.

System Manager:
A VMS user responsible for the overall operation of a VMS system. Responsibilities of the system manager include authorizing all users of the system, setting access requirements for all system resources, and running all procedures necessary to ensure the correct and timely operation of the system.

System Workspace:
A task workspace whose record definition is provided by ACMS, which provides three system workspaces. At run-time, ACMS fills in the contents of the system workspaces for each task selected by a terminal user. These workspaces, like other task workspaces, last only for the duration of a task instance.

See also workspace and task workspace.

SYSTEM-owned Set Type:
In DBMS, a set type owned by the SYSTEM record rather than by a record type you have selected. A SYSTEM-owned record has only one occurrence in the database, but can be the owner of many member record types. It allows unassociated record types to be used as entry points into the database.

See also member record type and owner record type.

Table:
See relation.

Tag Variable:
An optional variable in CDDL VARIANTS field description statements. The run-time value of the tag variable determines the current VARIANT.

See also VARIANTS field description statement.

Task:
A unit of work that performs a specific function and that a terminal user can select for processing. Every task belongs to a task group. Some tasks are defined in the task group they belong to; other tasks have separate task definitions. In either case, they are defined with the ACMS Application Definition Utility. The work of a task can be defined as a single processing step or a block step, which consists of a series of exchange and processing steps.

See also single-step task and multiple-step task.

Task Debugger:
A debugging tool, provided by ACMS, that is primarily for debugging multiple-step tasks that use procedure servers. The Task Debugger uses task group databases and procedure server images; it does not require application definitions, menu definitions, or a running ACMS system.

Task Group:
One or more ACMS tasks that have similar processing requirements and that are gathered together so they can share resources. A task group definition, created with the Application Definition Utility, defines the servers used by the tasks that belong to the group. It also defines other characteristics and requirements for the tasks in the group, such as workspaces, request libraries, and message files.

Task Group Database (TDB):
A run-time database containing information derived from task and task group definitions. The Task Debugger uses the TDB when debugging tasks; the Application Definition Utility uses the TDB when building an application database. ACMS also uses the TDB when a terminal user selects a task. The TDB is created as a result of building a task group definition with the Application Definition Utility.

Task Instance:
The occurrence of the processing of a task. Each selection of a task is a task instance. Every task instance is given a unique ID by the ACMS run-time system. ACMS processes a task instance in the manner prescribed by the definition of the task selected.

Task I/O:
The communication between a terminal user and a task instance. This communication can consist of VMS terminal I/O or TDMS requests.

Task Selection String:
In ACMS, the string of characters a terminal user types, in addition to the selection keyword or number, when making a selection from a menu.

Task Submitter:
Any authorized ACMS user who selects tasks for processing, provides input for that processing, and receives the results of that processing. Task submitters must also be authorized VMSusers. Synonymous with terminal user.

Task Workspace:
A workspace used mainly to pass information between steps in a multiple-step ACMS task. A task workspace is allocated when a terminal user starts a task and keeps its contents only for the duration of the task instance.

TDB:
See task group database.

TDMS:
See Terminal Data Management System.

Tenant Record:
Any DBMS record that participates in a set, whether a member or owner.

Terminal Control Subsystem:
A set of ACMS-controlled processes that control terminal user access to ACMS. The terminal control subsystem includes two types of processes: the command process or processes and the terminal subsystem controller.

Terminal Data Management System (TDMS):
A VAX product that uses forms to collect and display information on the terminal. TDMS provides facilities for interactively creating customized forms that have different kinds of field and text characteristics, as well as several video features (such as bolding and reverse video). TDMS provides data independence by allowing data used in an application to be separated from the application program. ACMS multiple-step tasks use TDMS services to manage terminal input and output.

Terminal Server:
The part of DATATRIEVE that gives you access to DATATRIEVE's interactive data management services.

Terminal Subsystem Controller:
The process in the terminal control subsystem that controls which terminals have access to ACMS.

Terminal User:
An ACMS user authorized to select tasks for processing.

Termination Procedure:
A procedure that runs when a procedure server process stops and that usually closes files or releases databases.

TEXT Data Type:
In DECgraph, a field value indicating that X values are to be considered as text rather than as numbers. Text may include numbers that are really names, such as years (1980, 1981), equipment names (VAX 730, 750), or fiscal quarters (1, 2, 3, 4).

See also NUMERIC data type.

Trace Facility:
The facility that helps you to debug a TDMS application by letting you monitor the action of a TDMS application program at run time. You can use Trace to:

- Trace the execution of a request at run time, including:
  - Transfer of data from a form to a program record
  - Transfer of data from a program record to a form
  - Values of control fields
- Trace TDMS calls, including:
  - Parameter values
  - Entry and exit time of each call

Transaction:
A set of operations on a recoverable resource such as a database. The operations in a transaction are treated as a group; either all of them are completed at once, or none of them is completed.

In DBMS and Rdb/VMS, a transaction groups a series of statements that perform a task.

- In DBMS, a transaction normally begins with a READY statement and ends with a COMMIT or ROLLBACK statement. However, a transaction may begin with any DML statement, other than READY, if the previous transaction in the run unit ended with a COMMIT statement that contained a RETAINING clause. DBMS transactions include only data manipulation operations.
- In Rdb/VMS, a transaction normally begins with START_TRANSACTION and ends with COMMIT or ROLLBACK. Rdb/VMS transactions can include data manipulation or data definition statements.

See also commit, quiet point, recovery, and rollback.

Trend Line:
In a scatter chart, a line that identifies the progression of the data for only the first set of Y values. In DECgraph, the TREND options icon leads to the icon choices: TREND and NO TREND.

Tuple:
See record.

Type:
A characteristic of each element in the CDD. Directories and subdictionaries are directory types, and there are several types of dictionary objects (for example, CDD$RECORD, DTR$DOMAIN, and DBM$SCHEMA).

Unique Name:
A designation assigned to a component, such as a task, that is used to identify that component within and across definitions.

Usage Mode:
In DBMS, the combination of the DML READY statement's allow mode and the access mode. It describes how a realm you have readied can be used. The eight usage mode combinations are:

BATCH UPDATE	BATCH RETRIEVAL
PROTECTED UPDATE	PROTECTED RETRIEVAL (default)
CONCURRENT UPDATE	CONCURRENT RETRIEVAL
EXCLUSIVE UPDATE	EXCLUSIVE RETRIEVAL

See also access mode and allow mode.

User Definition File:
A file, created and maintained with the User Definition Utility, that contains a list of users authorized to access ACMS.

User Definition Utility:
The ACMS tool for authorizing ACMS users and defining charateristics of those users.

User Name:
A designation assigned to a VMS user to identify that user. Also, the name a terminal user types to log into VMS and ACMS.

User Utility:
See User Definition Utility.

User Work Area (UWA):
In DBMS, a portion of memory assigned to your run unit that holds data to be transferred between your run unit and the DBCS. It holds data that is either going from your run unit to the database, or is coming from the data base to your run unit.

User Workspace:
A workspace, defined as an attribute of a task group, that holds information about a terminal user. A user workspace is created the first time a terminal user starts a task that references it. ACMS keeps a separate copy of a user workspace for each user and saves the contents of the workspace until the user exits from ACMS.

UWA:
See user work area.

Valid Request:
A TDMS request in the CDD with the following characteristics:

- The form and record definitions named in the request are stored in the CDD
- The record field and form field names used in mapping instructions are the same as those contained in the form and record definitions
- According to TDMS mapping rules, the following are compatible:
  - The data types of fields mapped to each other
  - The lengths of fields mapped to each other
  - The types (for example, simple, group, array) of fields mapped to each other

Validate Mode:
See validation.

Validation:
The process of checking data on entry to ensure that it meets preestablished requirements.

In DATATRIEVE and Rdb/VMS, for example, the VALID IF clause in the record definition sets criteria for validation of values entered for storage.

In TDMS and ACMS, the definition utilities, when in Validate mode, check that references to external definitions are correct before storing a definition in the CDD.

See also valid request and valid request library definition.

Value:
In DECgraph, a single piece of data. Y values must be numeric. X values can be either numeric or text.

Value Expression:
A symbol or string of symbols that you use to calculate a string or numeric value. When you use a value expression in a statement, Rdb/VMS or DATATRIEVE calculates the value associated with the expression and uses that value when executing the statement.

Variable:
A name associated with an expression whose value can change.

In DATATRIEVE, a variable is a value expression created in a DECLARE statement. For example, the following statement creates a variable, X, that can be assigned any two-digit numeric value:DECLARE X PIC 99.

VARIANTS Field Description Statement:
A CDDL statement defining a set of two or more fields that provide alternative descriptions for the same portion of a record. The function of the VARIANTS field description is similar to that of the REDEFINES clause in VAX COBOL and VAX DATATRIEVE.

Vertical Label:
In DECgraph, the name that identifies or describes the Y values. It appears along the left side of the screen from the bottom to the top. For example, in the Annual Reports data, the vertical label "Dollars" describes what type of data the Y values represent.

Video Attribute:
A characteristic of a TDMS form that provides one or more of the following special visual effects to an area of a form:

- Reverse video (light background, dark text)
- Bolding
- Blinking
- Underlining
- Double-height characters
- Double-width characters

View:
A subset of an Rdb/VMS database that includes any combination of fields and records from a single relation or from different relations. You form a view using the select or join operation. To the user, the results look like a single relation.

Views are most useful for two purposes:

- Restricting access to parts of the database
- Making the results of selections and joins permanent

See also view domain.

View Domain:
A special type of DATATRIEVE domain that allows you to select some (or all) fields in some (or all) records from one or more domains.

VMS:
The operating system on a VAX computer.

VMS Image:
See image.

VMS Process:
See process.

VMS User:
A person or account authorized by a VMS system manager to access a VMS system. A VMS user is assigned a user name, a password, a user identification code (UIC), a default directory, a default command language, quotas, limits, and privileges.

Wildcard Character:
A symbol, such as the asterisk, question mark, or percent sign, that you use in place of all or part of a file specification.

Word List Menu:
A DECslide menu that uses a list of words to present its choices. The object and change menus are examples of word list menus.

Word Slide:
Any DECslide slide composed of one or more words used to present information. A word slide can also include a graphics object or two.

See also graphics slide.

Work Area:
In DECslide, the space on your screen above the message line where you design your work. This area also displays the keypad diagrams and help text when you request them.

Workspace:
In ACMS, a buffer used to save variable context between steps and tasks, whose description is stored in the CDD. A workspace can also hold application parameters and status information. Workspaces are passed to step procedures as parameters. There are three types of workspaces: task workspaces, group workspaces, and user workspaces. ACMS provides record descriptions for three task workspaces, which are referred to as the system workspaces.

See also group workspace, task workspace, system workspace, and user workspace.

Workspace Symbol Module:
An object module, produced as a result of building a task group definition, that contains a main routine and debug symbol table used by the ACMS Task Debugger to examine workspaces. The object module must be converted into an executable image by the LINK command before the Task Debugger can use it.

X Axis:
The bottom horizontal edge of a chart.

X Data Type:
In DECgraph, the type of data used for the X values. If you use numbers, the X data type field must read NUMERIC. If you use text for the X values, the X data type must read TEXT.

X Unit:
In DECgraph, a field that indicates the multiplying factor for the X numeric values, ranging from E-30 to E+30. For example, if you choose the value "Thousands" for the X unit field, you can enter the number 1 in the X values to mean 1,000 or the number 15 to mean 15,000.

Y Axis:
The vertical left edge of a chart.

Y1-Y6 Legends:
In DECgraph, text that describes the data in the Y1-Y6 rows of values. The text you enter in the Y1-Y6 legends appears in the finished chart in the legend box.

Y Unit:
In DECgraph, a field that indicates the multiplying factor for the Y numeric values, ranging from E-30 to E+30. For example, if you choose the value "Thousands" for the Y unit field, you can enter the number 1 in the Y values to mean 1,000 or the number 15 to mean 15,000.