DATABASE APPROACH TO APPLICATION PROGRAM INTEGRATION FOR STATE COMMISSION USE

by

Jeffrey Shih
Computer Specialist

The National Regulatory Research Institute
2130 Neil Avenue
Columbus, Ohio 43210

October 1985

TABLE OF CONTENTS

FOREWORD

Many of our commission clients consider the development of "tools and techniques" as among the most useful products of NRRI's efforts. This report is offered in that light.

It describes a procedure NRRI developed for use on microcomuters to solve the problem of computer software with data that cannot be easily interchanged between two or more programs for lack of a consistent mechanism for transferring them. The thrust of the integration procedures shown here is the combining of data management and data analysis within a single framework.

I commend it to your attention and use.

Douglas N. Jones
Director
November 1985
Columbus, Ohio

1. INTRODUCTION

This report describes a procedure for integrating application programs. This procedure is developed by The National Regulatory Research Institute (NRRI) for use by state commissions on microcomputers. A problem that generally exists among users of computer software is that data cannot be interchanged between two or more programs due to the lack of a consistent mechanism for transferring them. This creates many instances where a large amount of data will have to be re-entered before it can be used. For example, suppose a set of future-year electric loads is calculated by an electric load forecasting program that has no graphics capability. This set of loads may consist of hundreds of values. If the hourly load shapes were desired then a separate program capable of generating these graphs would be needed. These hundreds of data would have to be entered all by hand to this graphics program if the format of the data file containing these loads does not match that specified by the program. It is apparent that the inability to manage and use data in an organized fashion imposes tremendous burdens on the users. Duplicate data entry becomes a must and may also create potential problems because of errors associated with re-entering the data and the need to maintain multiple sets of data files. State commissions process a large amount of information during the course of rate-making proceedings and therefore this problem could be very significant. While commercially available database management software would fulfill the data management role, they fall short of addressing the issue of data analysis.

The main purpose of the integration procedure described here is to help solve this problem by combining data management and data analysis within a single framework to enhance state commissions' problem-solving and decision-making capabilities. This arrangement provides a work environment in which one can view and/or change program inputs, run individual programs, and analyze program outputs in a systematic and consistent manner. While working in this environment, one can switch back and forth among various application programs and between programs and data quickly and efficiently. This is especially useful in situations where either the solution to a particular problem requires the use of several independent programs or an evaluation of alternatives calls for a "what-if" type of sensitivity analysis. This procedure is applicable to all utility sectors, and an example application was developed by the NRRI for use in the electric utility sector. It should be noted that the term "integration" refers to the framework for implementing the concept of combining the capabilities of individual programs in a systematic manner; it is not intended to imply that application programs are simply united in an all-in-one program structure.

The integration scheme described here is based on the database approach to data processing system development. The next section of this report contains a description of this database approach. Briefly, this approach calls for the construction of a database containing a set of logically coherent information about a well-defined subject. With this method, it is assured that each piece of data contained in the database carries the same meaning for all who use it. It is then possible for the same data item to be shared by different programs for different purposes; that is, data can, in effect, be interchanged among various application programs. This is the most important step towards integration. Once the database is set up and its structure defined, the next step is to provide a facility for application programs--data analysis tools--to access the database. This facility, also known as the application program interface, enables an application program to move data in and out of the database regardless of the program's logic and modeling algorithms.

According to this integration procedure, a typical integrated system will have three components. They are:

(1) a database which contains both the application programs' input and output, and a database manager which is a program that performs all operations necessary to maintain the database;

(2) a library of application programs that perform the computations needed to transform program input to program output, and an application program manager which is a program that maintains this program library; and

(3) a control program which serves as the user interface to the integrated system.

Figure 1 shows a diagram of those components of a typical integrated system.

The organization of the report is as follows. Section 2 describes the database (data management) component of an integrated system. Included are discussions on the database approach to applications integration and the database management system that manages the database. Section 3 deals with the data analysis component of the integrated system and contains a description of the application program interface facility. Section 4 details a sample application of this integration procedure. Three of NRRI's microcomputer programs are used in this demonstration. They are an electric load analysis program (LOAD), an electric production cost simulation program (PROC), and an electric cost-of-service program (COST). Section 5 contains some concluding remarks regarding the usefulness of this integration procedure.

2. THE DATABASE

This section has two parts. In the first subsection, a description of what a database is and of how it can be applied toward applications integration is given. The second part of this section deals with the software that is used to manage databases--the database management system.

The Database Approach to Applications Integration

A database is a shared resource containing information about some subject. It may or may not be computerized. Normally data are organized into three basic units: files, records, and fields. Each file contains a number of records. Each record in turn is a grouping of several items of information that go together. Each such separate item of information in a record is called a field. An example of a nonautomated database is the card catalogs in a library. The catalog itself is the file, which contains a card (record) for each book in the library. Each card includes several fields of specific information, such as the title of the book, the author's

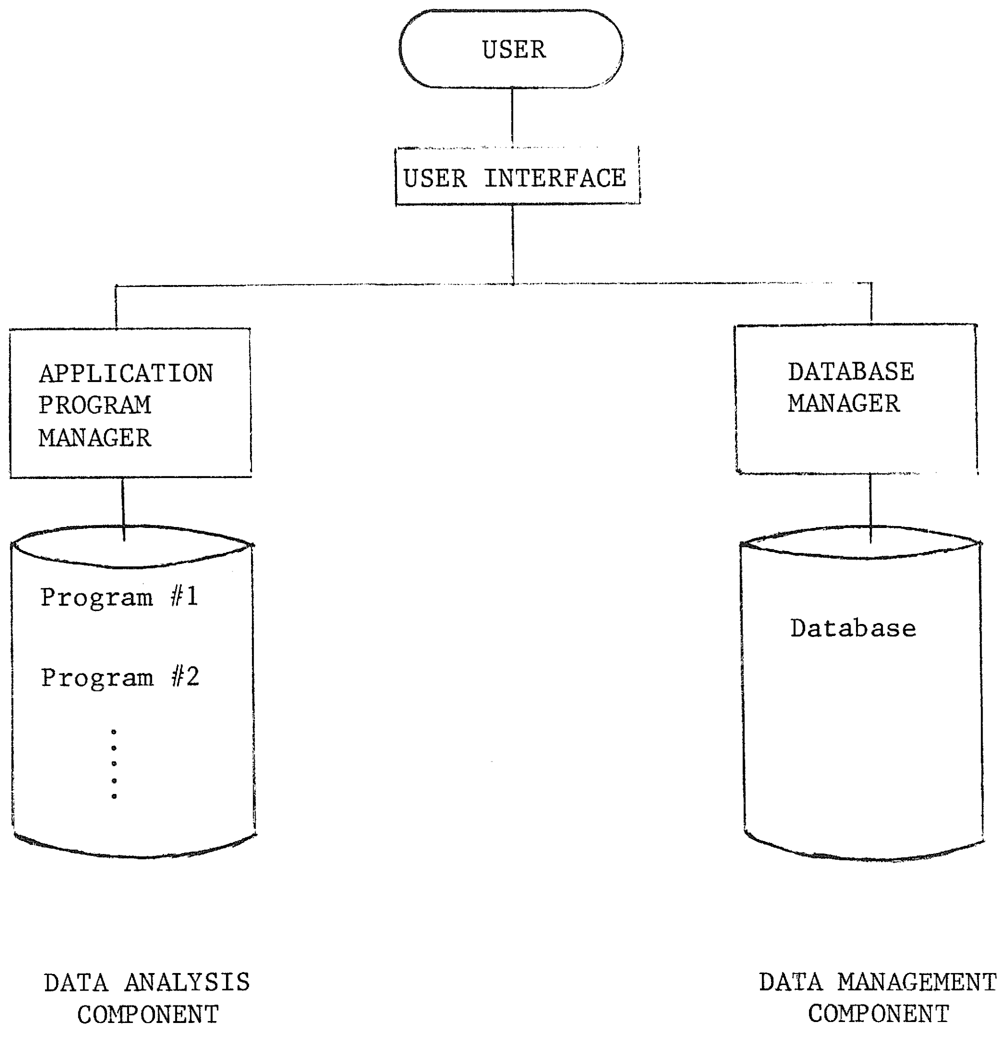

Figure 1. Components of a typical integrated system

name, and the subject matter of the book. Several such card catalogs (title catalog, author catalog, and subject catalog, for instance) comprise the database. In this example each card appears three times, once each in the title, author, and subject catalogs. In a computerized database, however, only one copy of each card is kept on disk.

A computerized database is often viewed as one or more rectangular tables of rows and columns. The library card catalog, if stored electronically, would look like the following:

	TITLE	AUTHOR	SUBJECT
1.	Managing Databases	M. Gorman	Database
2.	Power Systems Engineering and Mathematics	U. G. Knight	Power Systems
3.	Reliability and Risk Analysis	N. J. McCormick	Reliability

This table has three rows (records). Each row gives information about a book. The table also has three columns (fields). Each column contains information about one property (attribute) of the entity represented by a row. Each record is assigned a record number and each field has a name (column heading). The software employed to manage the database should enable one to search for a book by any one of the three fields, thus eliminating the need for duplicate copies of each card.

Having described what a database is, the remainder of this subsection provides a discussion of how the database concept can be applied toward applications integration. The traditional approach to data processing system development has as its cornerstone the programs. That is, systems are created of programs that read and update data, perform computations, and print results. Fundamentally these programs are written on the basis of output needs. The output is finalized first and input specifications are defined next, then the program is written to transform input to output. Very often the data are stored in formats that are most efficient for the data processing required of the program. Consequently all data semantics--which are rules for meaning, validity, and usage--are defined in terms of the program logic. It is obvious that data and program are definitely interdependent as each is merely an extension of the other. One may say that the data is "welded" to a specific program and as such is not likely to be useful for other programs.

Still one could argue that applications integration is attainable by means of an all-in-one product that each analysis function is but part of the whole. The data interchange problem is indeed addressed by this method since an internal, standard mechanism for transferring data is required in order for the product to function. The user, however, must stay within the boundaries of the built-in function. This could be quite difficult as the need to handle ad hoc situations eventually will go beyond the provided facilities. Furthermore, the closed architecture exhibited by such a system presents an additional roadblock as reprogramming is always needed to add new analysis functions.

With the database approach, however, data is no longer an extension of a program. It is rather a set of logically coherent information about a well-defined subject. Records have well-defined meaning within the context of the database and fields have well-defined meaning within the context of the record definition. In short, each piece of data carries the same meaning for all who use it. In contrast to the traditional approach where data is application program dependent, the same data item could be used by different programs for different purposes. Data can, in effect, be interchanged among various application programs. The software employed to manage the database, rather than individual application programs, is now responsible for the codification of and the adherence to data semantics. It is clear that the database approach to data processing system development provides an ideal framework for implementing applications integration.

An integrated system developed using this database method has an open architecture. New data analysis programs can always be added to handle ad hoc situations. In addition, a high level of consistency in handling data from various sources can be attained with this approach because data semantics are defined at the database level, not the end-use level. This benefit is of great significance if the domain of applications integration covers different functional divisions within an organization.

The Database Management System (DBMS)

A database management system (DBMS) is a software tool employed to manage computerized databases. It is a program or set of programs that provides a framework for creating, editing, and maintaining collections of data for use by different application programs. A DBMS is often thought to be simply a means to store and organize data. A common metaphor used is that of an electronic filing cabinet. Filing, however, is only one aspect of a DBMS's functions. It must also be able to present the data upon demand in a variety of formats and from a variety of viewpoints. That is, it must have a query facility for making individual requests for information from the database and a reporting facility for producing formatted listing of selected data.

For the purpose of storing and organizing data in the database component of the current study, the NRRI developed a computer program named DBMGR. DBMGR permits one to create, access, and update database files that have the same format and structure as those of a commerical product--dBASE III from Ashton-Tate. dBASE III is a fully-featured relational database management system that runs on an IBM PC or PC/XT. A relational database is one in which no implicit relationship (such as parent/child) exists between the files. Instead, the files are related by having a field in one file that is the same as a field in another file. In other words, relational databases rely on identical fields to relate items in one table to those in another. A DBMS that manages databases using this relational model is called a relational DBMS.

Functionally DBMGR is a subset of dBASE III in that one can access and modify the same database interchangeably with either program, but that the query and reporting facilities available from dBASE III are absent in DBMGR. Users who are familiar with the dBASE III program should find that DBMGR closely resembles dBASE III in terms of user interfaces, such as similar screen displays and consistent keyboard definitions in the editing mode. The remainder of this section contains a description of the data structure supported by the DMBGR program and its operations. The discussion holds true for dBASE III as well.

Like dBASE III and most DBMSs that run on microcomputers, DBMGR organizes data into two-dimensional tables of rows and columns. Each row is a record and is assigned a record number. Each column is a field and has a distinct name, called field name, up to ten characters long.

The following illustration gives a conceptual view of such a table.

	Field 1	Field 2	Field 3	...
	NAME	ADDRESS	PHONE-NO	
Record 1				
Record 2				
Record 3				
.				
.				
.				

In this example, NAME is the field name of the first column; ADDRESS, that of the second column; and so forth.

All items in a column (field) are of the same data type. A data type is a high level representation of data as seen by the user, which has a corresponding internal binary form understood by the computer. Data types allow one to write programs using data representations with which he is most compatible. These high level representations are maintained internally in a binary format processed by the computer. Four data types are provided by DBMGR: Character, Date, Logical, and Numeric. The following is a brief discussion of these data types.

Character data type--It is used to store any printable character that can be entered from the keyboard. It is often convenient to use the character data type for numbers such as telephone numbers and zip codes which will not be used in calculations. The maximum size of a character data type is 254.

Date data type--It is used to store dates in the mm/dd/yy format. The size of a date data type is always 8.

Logical data type--It is used to keep track of things like paid/unpaid, male/female, and similar items that can only be one of two things. A logical data type is always 1 byte long.

Numeric data type--It is used to represent integers or decimal quantities that will undergo computations. The size of a numeric data type is the number of digits it can hold (the decimal point, if any, counts as one digit).

DBMGR uses these data types as the building blocks to construct database file structures matching the body of data that they are intended to respresent. Below is an example structure of a database file containing shipping information of, say, a mail-order company.

```
Structure for database :  example.dbf
Number of data records :      30
Date of last update    :    08/08/85
  Field        Field name        Type          Width      Dec
    1          SHIP-TO           Character       20        0
    2          DATE-SHIP         Date             8        0
    3          PRODUCT           Character       30        0
    4          QUANTITY          Numeric          5        0
    5          AMOUNT-DUE        Numeric          7        2
    6          INV-PAID          Logic            1        0
**Total**                                        72
```

This display shows the file name, the number of records in the file, when the file was last updated, and specifications of all the fields of a record. For each field these specifications are the name of the field (Field name), its data type (Type), size of the field (Width), and decimal places, if any, of the field (Dec). Each record in this example database file has six pieces of information indicating the customer receiving the shipment, date the shipment was made, what was shipped and how much, total amount due from the customer, and the payment status, respectively.

There are six options in the DBMGR that would enable one to create, access, and update database files. They are APPEND, BROWSE, CREATE, DISPLAY, EDIT, and USE. The CREATE option allows one to build a file structure similar to the one shown above. While in the CREATE mode, one can interactively enter appropriate information regarding the field name, data type, and so forth. DBMGR has a built-in facility to check the validity of the information entered from the keyboard. For example, if one enters a field name that has already been assigned to a previous field definition, DBMGR would display the "duplicate field name" message and would prompt the user for a different field name instead.

The APPEND option is used to add new data records to an existing database file. A data entry form will be displayed by the program for one to fill in appropriate information for each added record. This entry form is generated according to the structure of the database file under consideration. Using the same example given above for the mail-order company, the entry form would look like the following illustration. The database

```
Record No.          31          File name:  example.dbf

SHIP-TO             ________________________
DATE-SHIP           ________________________
PRODUCT             __/__/__________________________________
QUANTITY            ________________________________________
AMOUNT-DUE          _______
INV-PAID            ___.___
                    _
```

file name and the record number are shown at the top of the entry form. Field names are listed on the left-hand side of the form, with corresponding blank spaces on the right-hand side. The size of blank spaces is determined by the width of each field specified in the file structure. Note that two slashes "/" are used for Date data type to conform to the mm/dd/yy format. Note also the presence of a decimal point for field AMOUNT-DUE, which represents a decimal quantity (see the example file structure given above).

The program will check each user input to verify its validity. For example, if one enters a string of characters whereas the field is of a Numeric data type, this invalid entry will be rejected by the DBMGR program. As another example, if one enters 08/40/84 or 15/02/85 for a Date data field, similar action will be taken by the program to block such an erroneous entry.

BROWSE option is used to display the content of the entire database file under consideration. While in this mode, field names are listed across the top of the display screen, with individual records displayed in the remainder of the screen. DBMGR allows one to use various cursor movement keys in the numeric keypad area of the keyboard to move through the entire file. For example, pressing the PgDn key will bring into view the next set of records that the display screen could not hold previously.

EDIT option is used to provide the user with a full screen editing environment for data update purposes. Like the BROWSE option, one can use cursor movement keys to move through the entire file in order to locate the appropriate record and field desired for data update. The same built-in facilities for data validity checking described under the APPEND option also apply here. Any invalid information entered while in this EDIT mode will be identified and rejected for re-entry. Upon completion and leaving the EDIT mode, all changes made will be permanently saved on disk. The USE option is used for one to gain access to a particular database file. Unlike dBASE III which permits ten database files to be accessed all at one time, DBMGR uses one file at a time. When this option is invoked, the user's only input requested is the name of the database file to be accessed by the program.

Altogether, these six options provided by DBMGR permit one to store and organize data in a database. Files in the database have the same format and structure as those of dBASE III, and as such one can substitute dBASE III for DBMGR for more sophisticated operations such as making ad hoc queries into the database.

3. APPLICATION PROGRAMS

The first part of this section details an interface facility that allows application programs to access the database component of an integrated system described in the preceding section. A utility program named APMGR that manages the collection of application programs using this interface facility for input/output preparations is described in the second part of this section.

Application Program Interface

An application is some instance when one or more database files are used for some intended data analysis. An application requires the presence of a program which controls the computer--it provides instructions for necessary operations such as input, output, and computations. This program, called an application program, is generally written by a program developer in a high-level programming language such as Fortran or BASIC. An access facility must be provided in order for an application program to use data in the database. This access facility, known as the application program interface, makes the internal structure of the database known to application programs. This capability of moving data between database files and application programs is crucial in implementing the integration procedure.

The interface described here allows application programs to access database files created by DBMGR (and dBASE III). It consists of several routines that provide the necessary input/output operations for an application program to use data in the database. These routines are based on the internal structure of the database described below.

Each database file stored internally by the DBMGR program is composed of a file header and data area. The file header is a block of data which contains complete information about the file structure. The data area holds the actual content of all records in the file. The layout of the file header is described first.

The first 32 bytes (character) of the file header contain the general information about the database file under consideration, such as how many records it contains and the date it was last updated. Specifically these 32 bytes of data convey the following meanings:

Byte	Content
0	A hexadecimal number indicating the file type. It is always 03 in this study.
1	A hexadecimal number representing the last two digits of the year when the file was last updated.
2	A hexadecimal number representing the month when the file was last updated.

Byte	Content
3	A hexadecimal number representing the day when the file was last updated.
4	The first byte of a 4-byte hexadecimal number representing the total number of records in the file.
5	The second byte of a 4-byte hexadecimal number reprsenting the total number of records in the file.
6	The third byte of a 4-byte hexadecimal number representing the total number of records in the file.
7	The fourth byte of a 4-byte hexadecimal number representing the total number of records in the file.
8	The first byte of a 2-byte hexadecimal number representing the location (offset) of the data area.
9	The second byte of a 2-byte hexadecimal number representing the location (offset) of the date area.
10	The first byte of a 2-byte hexadecimal number representing the length of the record.
11	The second byte of a 2-byte hexadecimal number representing the length of the record.
12-31	Unused bytes filled with hexadecimal number 00 (zero-filled).

With this block of data and some necessary conversion from the hexadecimal to the decimal numbering system, information such as the record length is readily known to an application program. For example, suppose byte 10 and byte 11 contain 7D and 00, respectively, then the record length can be determined as:

$$\underbrace{\overset{16^3}{\underline{}0}\ \overset{16^2}{\underline{}0}}_{\text{2nd byte}} \quad \underbrace{\overset{16^1}{\underline{}7}\ \overset{16^0}{\underline{}D}}_{\text{1st byte}}$$

$$\begin{aligned}\text{record length} &= 0\text{x}16^3 + 0\text{x}16^2 + 7\text{x}16^1 + 13\text{x}16^0 \\ &= 0{+}0 + 112 + 13 \\ &= 125 \ \text{decimal,}\end{aligned}$$

noting that hexadecimal D is equal to 13 decimal.

The remaining portion of the file header contains information about data fields that constitute the database file under consideration. Each field definition uses 32 bytes in the file header, as described below.

Byte	Content
0-9	Field name expressed in ASCII (American Standard Code for Information Interchange) characters. The maximum length of a field name is 10 characters; any field name of less than 10 characters long is left justified and zero-filled.
10	A hexadecimal number (00) used as a separator.
11	Field data type expressed in ASCII, it is one of the four types supported: C (Character), D (Date), L (Logic), and N (Numeric).
12-15	Unused bytes.
16	A one-byte hexadecimal number representing the size of the field.
17	A one-byte hexadecimal number representing the number of decimal places of a decimal quantity.
18-31	Unused bytes.

This 32-byte pattern repeats for each and every field defined in the database file.

The remainder of the database file is the data area. Hexadecimal constants 0D and 1A mark the beginning and the end of this data area, respectively. All data records are stored in between these two markers, following the layout described below.

1. Each data record is preceded by a hexadecimal constant 20 (a space) and there is no record terminator.
2. Data fields are packed with no field separators.
3. Data types are stored in ASCII format. The Date data types are stored in an 8-byte, YYYYMMDD format, such as 19850831 for August 31, 1985.

The appendix to this report contains several routines that are written according to these file structure specifications. These routines are provided so that one can custom-tailor application programs for his own applications integration.

Library of Application Programs

Recall that the database approach to the applications integration provides an assurance that the same data item can be shared by different programs for different purposes. This assurance is valid only in situations where all data semantics are defined at the database level, not at the program level. Consequently an application program of which the input and output operations conform to the specifications laid out by the database is said to be suitable for implementation in the integration environment described in this report. This requirement is necessary in order to employ the database approach to integration.

Application programs that are written following this approach are collectively maintained in a program library through a computer program. This utility program, named APMGR, is developed by the NRRI for this study. APMGR creates and maintains a list of programs that are suitable for implementation in this integration environment. With the aid of this program, new programs can be added to and out-dated programs can be deleted from this library of application programs. This library of programs forms the data analysis part of the integrated system. In addition to its library maintenance duty, APMGR also allows one to run any of the programs contained in the library without having to go back to the operating system. One can therefore switch back and forth among various application programs as needed quickly and efficiently.

4. A SAMPLE APPLICATION

This section describes a sample application of the integration procedure. Three of NRRI's microcomputer programs are used. They are an electric load analysis program (LOAD), an electric production cost simulation program (PROC), and an electric cost-of-service program (COST). The first part of this section deals with the first step of the integration procedure--the construction of a database. The second subsection contains a discussion of the implementation of the application program interface--the second step of the integration procedure. The last part of this section describes the operation of this integrated system.

The Construction of a Database

The first step of this integration procedure calls for the construction of a database. This database is to contain both the application programs' input and output. For the purpose of demonstrating a sample application, a database is set up to hold information about an electric utility system. Specifically, this sample database contains data on the utility's load profile, the generating units, and costs of providing services to its customers. This database is included here only as example material and is not intended to be an actual representation of an electric utility company. It is used to illustrate the applications integration procedure described in this report. This sample database is composed of nine database files. They are described below.

Database File: PLANT.DBF

This database file contains generating unit data of a hypothetical electric utility. The structure for this database file is displayed in figure 2.

The definition of individual data fields is given below.

Field 1 -- A unique generating unit identification code number.

Field 2 -- The unique name of the generating unit of up to ten characters long. It may contain letters A through Z, numbers 0 through 9, and special symbols.

Field 3 -- The primary fuel used by the unit for generation. It can be one of the following: coal, nuclear, oil2 (for No. 2 oil), oil6 (for No. 6 oil), gas (for natural gas), and gasoline.

Field 4 -- Numerical codes are assigned to units to identify their expected duty cycle. The commitment order is controlled by this code assigned to units. Operating type 1 refers to base load units, types 2 through 4 refer to intermediate load units with type 2 loaded first and type 4 last, and type 5 refers to peaking units.

```
Structure for database:  PLANT.DBF
Number of data records:  0
Date of last update   :  07/01/85
Field      Field name    Type         Width     Dec
    1      UNIT_CODE     Character        3
    2      UNIT_NAME     Character       10
    3      FUEL_TYPE     Character        4
    4      OP_TYPE       Character        1
    5      FOR           Numeric          6       2
    6      FUEL_COST     Numeric          7       2
    7      VAR_OM        Numeric          5       2
    8      FIX_OM        Numeric          6       2
    9      HEAT_CONT     Numeric          6       2
   10      SO2_EMISON    Numeric          5       2
   11      NOX_EMISON    Numeric          5       2
   12      MAINTENANC    Numeric          6       2
   13      CAP_LVL1      Numeric          7       2
   14      CAP_LVL2      Numeric          7       2
   15      CAP_LVL3      Numeric          7       2
   16      CAP_LVL4      Numeric          7       2
   17      HR_LVL1       Numeric          8       2
   18      HR_LVL2       Numeric          8       2
   19      HR_LVL3       Numeric          8       2
   20      HR_LVL4       Numeric          8       2
**Total**                               125
```

Figure 2. Structure for database file PLANT.DBF

Field 5 -- Unit forced outage rate in per cent.

Field 6 -- Unit fuel cost in ¢/MBtu.

Field 7 -- Unit variable operation and maintenance cost in $/MWh.

Field 8 -- Unit fixed operation and maintenance cost in $ per year per unit.

Field 9 -- Fuel heat content in MBtu/ton for solid fuels or MBtu/bbl for liquid fuels.

Field 10 -- Unit SO_2 emissions in lbs per MBtu of fuel burn.

Field 11 -- Unit NO_x emissions in lbs per MBtu of fuel burn.

Field 12 -- Unit scheduled maintenance in days.

Field 13 -- The heat rate curve of each unit is represented at a maximum of four capacity levels: MW1, MW2, MW3, and MW4. Each unit can be brought on line in a maximum of four blocks of capacity: MW1, MW2-MW1, MW3-MW2, and MW4-MW3. Unit capacity (in MW) at MW1 level is reported in this field.

Field 14 -- Unit capacity value in MW at the second level. If a unit is to be loaded to its full capacity in one step, then the capacity value reported here should be the same value as reported in field 13.

Field 15 -- Unit capacity value in MW at the third capacity level.

Field 16 -- The maximum dependable capacity of the unit in MW.

Field 17 -- Unit heat rate value at the first capacity level in Btu/kWh.

Field 18 -- Unit heat rate value at the second capacity level in Btu/kWh.

Field 19 -- Unit heat rate value at the third capacity level in Btu/kWh.

Field 20 -- Unit heat rate value at the maximum capacity level in Btu/kWh.

Database File: LOAD.DBF

This database file contains data on the utility's hourly load. Figure 3 displays the file structure. The first field contains day type identification information. A day type means either (1) a calendar day such as May 1, 1984; or (2) a representative day if you aggregate the days that exhibit similar hourly load patterns to form a fictitious day as their representative, such as "typical weekday" and "typical weekend day." This latter approach is usually used to reduce the computer memory requirement and to ease the computational burden when hourly production simulation is desired. The information contained in this field could be "Weekday" for a typical weekday or "1/28/83" for a particular calendar day. The second field contains the total number of days in a given day type under consideration. For example, if one chooses to use two representative day types to represent a weekly load cycle: a typical "weekday" and a typical "weekendday"; then he would have five days under the "weekday" and two days under the "weekendday." The number for a calendar day should always be one (1). Other fields of the file contain the load values in MW. For instance, field 12 contains the load at the 10th hour of a day.

Database File: AVELOAD:DBF

This database file is used to contain the average hourly load calculated by the LOAD program. The structure for this file is the same as that described for LOAD.DBF (see figure 3).

Database File: OPCOST.DBF

This database file is used to contain individual generating unit operating and cost characteristics for a given time step of analysis, as a result of simulations performed by the PROC program. The structure for this file is shown in the top part of figure 4. The individual data fields defined in this file have the following meaning.

```
Structure for database:  LOAD.DBF
Number of data records:  0
Date of last update   :  07/01/85
Field       Field name   Type          Width     Dec
    1       TYPE_ID      Character        10
    2       FREQ         Numeric           3
    3       HR1          Numeric           5
    4       HR2          Numeric           5
    5       HR3          Numeric           5
    6       HR4          Numeric           5
    7       HR5          Numeric           5
    8       HR6          Numeric           5
    9       HR7          Numeric           5
   10       HR8          Numeric           5
   11       HR9          Numeric           5
   12       HR10         Numeric           5
   13       HR11         Numeric           5
   14       HR12         Numeric           5
   15       HR13         Numeric           5
   16       HR14         Numeric           5
   17       HR15         Numeric           5
   18       HR16         Numeric           5
   19       HR17         Numeric           5
   20       HR18         Numeric           5
   21       HR19         Numeric           5
   22       HR20         Numeric           5
   23       HR21         Numeric           5
   24       HR22         Numeric           5
   25       HR23         Numeric           5
   26       HR24         Numeric           5
**Total**                                134
```

Figure 3. Structure for database file LOAD.DBF

```
Structure for database:  OPCOST.DBF
Number of data records:  0
Date of last update   :  07/01/85
Field      Field name   Type         Width     Dec
    1      UNIT_CODE    Character        3
    2      PERIOD_NO    Character        2
    3      EL_ENERGY    Numeric          8
    4      TH_OUTPUT    Numeric          6
    5      CAP_FACTOR   Numeric          5       1
    6      SO2          Numeric          6
    7      NOx          Numeric          6
    8      FUEL_COST    Numeric          6
    9      OM_COST      Numeric          6
   10      OTHER_COST   Numeric          6
   11      TOTAL_COST   Numeric          8
   12      AVE_COST     Numeric          6       2
**Total**                               69
```

```
Structure for database:  SUMMARY.DBF
Number of data records:  3
Date of last update   :  07/01/85
Field      Field name   Type         Width     Dec
    1      PERIOD_NO    Character        2
    2      HOURS        Numeric          5
    3      TOTAL_CAP    Numeric          8
    4      PEAK_LOAD    Numeric          6
    5      MIN_LOAD     Numeric          6
    6      TOTAL_ENY    Numeric          8
    7      TOTAL_GEN    Numeric          8
    8      UNSERV_ENY   Numeric          8
    9      SO2          Numeric          8
   10      NOx          Numeric          8
   11      FUEL_COST    Numeric          8
   12      OM_COST      Numeric          8
   13      OTHER_COST   Numeric          8
   14      TOTAL_COST   Numeric          8
   15      AVE_COST     Numeric          6       2
   16      LOLP         Numeric          7       4
**Total**                              113
```

Figure 4. Structures for database files OPCOST.DBF and SUMMARY.DBF

Field 1 -- A unique generating unit identification code number. This is the same field defined in file PLANT.DBF (see figure 2). One can therefore join these two database files together through this identical field.

Field 2 -- The n-th time step of production simulation, where n is between 00 and 12. Currently the PROC program allows a maximum of 12 time steps of analysis to be specified by the user. A "01" in this field indicates the first time step of analysis, a "02" for the second time step, and so forth. A "00" is used to represent the entire study period, i.e., the summary over individual time steps.

Field 3 -- Unit electric energy generation in MWh.

Field 4 -- Unit thermal energy output in billion Btu.

Field 5 -- Unit capacity factor in per cent

Field 6 -- SO_2 emissions in tons.

Field 7 -- NO_x emissions in tons.

Field 8 -- Unit fuel cost in thousands of dollars.

Field 9 -- Unit operations and maintenance cost in thousands of dollars.

Field 10 -- Unit cost in "Other" category in thousands of dollars. It can be used to represent costs associated with pollution control devices, for example.

Field 11 -- Unit total cost in thousands of dollars. It is the sum over fields 8 through 10.

Field 12 -- Unit average cost of producing electricity in mill/kWh.

Database File: SUMMARY.DBF

This database file is used to contain production simulation results relevant to the generating system as a whole. The bottom part of figure 4 shows the structure for this file. The definition of individual data fields is given below.

Field 1 -- The n-th time step of simulation. This is the same field defined in the previous file--OPCOST.DBF.

Field 2 -- Total number of hours in this time step of analysis.

Field 3 -- Total generating system capacity in MW.

Field 4 -- Peak demand in MW.

Field 5 -- Minimum demand in MW.

Field 6 -- Total electric energy (GWh) demanded of the generating system.

Field 7 -- Total electric energy generation (GWh) delivered by the generating system.

Field 8 -- Unserved energy in GWh.

Field 9 -- SO_2 emitted by the generating system in tons.

Field 10 -- NO_x emitted by the generating system in tons.

Field 11 -- Fuel cost in thousands of dollars.

Field 12 -- Operations and maintenance cost in thousands of dollars.

Field 13 -- Other cost in thousands of dollars.

Field 14 -- Total cost in thousands of dollars. It is the sum over fields 11 through 13.

Field 15 -- Average cost of producing electricity, in mill/kWh, for the generating system as a whole.

Field 16 -- The lost-of-load probability of the generating system.

Database File: CUSTOMER.DBF

This database file contains data on the characteristics of the utility's classes of service. The structure for this database file is shown in the top part of figure 5. The first field contains a customer class identification code. The same identification code will be used in another database file (see CLS-ROR.DBF at the end of this subsection) so

```
Structure for database:  CUSTOMER.DBF
Number of data records:  0
Date of last update   :  07/01/85

Field       Field name   Type          Width      Dec
    1       CUSTOM_ID    Character         2
    2       CLASS_NAME   Character        30
    3       CUSTOM_NUM   Numeric          10
    4       ENERGY       Numeric          15
    5       COINC_PEAK   Numeric          10
    6       NONCO_PEAK   Numeric          10
    7       MON1_PEAK    Numeric          10
    8       MON2_PEAK    Numeric          10
    9       MON3_PEAK    Numeric          10
   10       MON4_PEAK    Numeric          10
   11       MON5_PEAK    Numeric          10
   12       MON6_PEAK    Numeric          10
   13       MON7_PEAK    Numeric          10
   14       MON8_PEAK    Numeric          10
   15       MON9_PEAK    Numeric          10
   16       MON10_PEAK   Numeric          10
   17       MON11_PEAK   Numeric          10
   18       MON12_PEAK   Numeric          10
**Total**                                198


Structure for database:  ACCOUNT1.DBF
Number of data records:  0
Date of last update   :  07/01/85

Field       Field name   Type          Width      Dec
    1       CAT_NO       Character         3
    2       CAT_TITLE    Character        50
**Total**                                 54


Structure for database:  ACCOUNT2.DBF
Number of data records:  0
Date of last update   :  07/01/85

Field       Field name   Type          Width      Dec
    1       CAT_NO       Character         3
    2       ACCOUNT_NO   Character         6
    3       ACT_NAME     Character        40
    4       AMOUNT       Numeric          10
    5       ALOC_ID      Character        30
**Total**                                 90
```

Figure 5. Structures for database files CUSTOMER.DBF, ACCOUNT1.DBF, and ACCOUNT2.DBF

that these two files can be linked together by the DBMS. The second field contains a unique name of the service class under consideration. Field 3 represents the total number of customers in a given class of service and field 4 contains information about the energy consumptions, in MWh, by this customer class. Field 5 contains data on class coincident peak demand measured in MW and the class noncoincident peak demand in MW is reported in field 6. Fields 7 through 18 represent twelve class monthly peaks measured in MW, respectively.

Database File: ACCOUNT1.DBF

This database file contains descriptions of the various items of utility plant in service and operations and maintenance expenses. Some examples are steam production plant, transmission plant, depreciation reserve, and steam operation expense. The structure for this file is displayed in the middle part of figure 5. This file contains only two data fields. The first field is an item category identification code and the description of this item is contained in the second field. The category identification code is used to provide a link between this database file and another database file named ACCOUNT2.DBF (see the next file description).

Database File: ACCOUNT2.DBF

This database file contains data on the various items of utility plant in service and operations and maintenance expenses. The bottom part of figure 5 shows the structure for this file. The first field is an item category identification code. This is the same field defined in the ACCOUNT1.DBF. The second field contains the account number for the item under consideration. The descriptive title and the dollar value of this account are represented in fields 3 and 4, respectively. The last field contains the allocation keyword used by the COST program to spread the cost among customer classes.

Database File: CLS-ROR.DBF

This database file is used to contain the summary cost allocation results produced by the COST program. Figure 6 shows the structure for this file. The first field is a customer class identification code. This is the same field defined in the CUSTOMER.DBF file. Information about the characteristics of a particular class of service contained in the CUSTOMER.DBF file can be made available to this file through this common data field. Operating revenues and operating expenses as allocated to various classes of service are contained in fields 2 and 3, respectively. Class net operating income is contained in the next data field. Class allocation of utility rate base is represented in field 6, and class rate of return is contained in the last field of this file.

Altogether, these nine database files in this example database represent data about an electric utility. These files serve as inputs to, as well as outputs from, the three application programs mentioned earlier. The structures for these files described above were constructed using the

```
Structure for database:  CLS-ROR.DBF
Number of data records:  0
Date of last update   :  07/01/85

Field      Field name    Type         Width      Dec
    1      CUSTOM_ID     Character        2
    2      TOT_OP_REV    Numeric          9
    3      TOT_OP_EXP    Numeric          9
    4      NET_OP_INC    Numeric          8
    5      RATE_BASE     Numeric          9
    6      RT_OF_RTN     Numeric          5        2
**Total**                                43
```

Figure 6. Structure for database file CLS-ROR.DBF

DBMGR program. The body of data can be entered according to these structures using either DBMGR or dBASE III program. Once the database is constructed, the next step is to implement the application program interface such that the data contained in the database can be used directly in application programs. This is discussed next.

The Implementation of Application Programs

A sample database has been constructed for this demonstration. The next step is to implement application programs to form an integrated system. For the purpose of demonstration, this integrated system is intended to perform analyses in the following areas: utility load, electricity production cost simulations, and class cost of service. Three existing microcomputer programs available from the NRRI could provide these intended functions. These three programs are an electric load analysis program (LOAD), an electric production cost simulation program (PROC), and an electric cost-of-service program (COST). They are therefore adapted for this sample application of the integration procedure.

Recall that an application program is suitable for implementation in the integrated system if its input/output is prepared using the program interface facility described in the previous section. None of the three programs uses this interface facility so some modifications are needed. Note that no such modifications would be required if one is to write new application programs, provided that the program interface facility is used for input/output operations.

The modifications are in some sense very localized since they are needed only in the input and some selected output portions of a program. For each of the three programs mentioned above, the input routine was first identified. This input function was then replaced by the interface facility's reading routine. A listing of this routine can be found in the appendix. As for output preparations, the interface facility's writing routine was used in places where output data are expected to be written to the sample database. Original program output routines such as writing to the display screen or to a printer are not affected. The appendix also contains a listing of the writing routine provided by the interface facility. After these modifications were made, programs were ready to become part of the integrated system. The APMGR program described earlier was used to add these programs to the application program library. This sample integrated system is now functional. A schematic of this system is depicted in figure 7 in which the database file usage is also shown.

The Operation of the Integrated System

As can be seen in figure 7, the two major components--the database and application programs--of the integrated system are coordinated by a program named AI. This program serves solely as the user interface to the integrated system. Upon invoking this program, one is provided with three options: (1) to access the database, (2) to access application programs, and (3) to end the operation of the integrated system. The first two options are described below.

Choosing option number one will invoke the DBMS program, and one will be given more options regarding operations on the database. As mentioned earlier, the database holds both the application programs' input and output. At this point the DBMS has the control over the integrated system, and all the functions provided by the DBMS are accessible to the user. One can, for example, edit data contained in one database file which serves as input to a particular application program; or he can view/analyze data in another file containing output from another application program. The user can exit this database component anytime as is desired; and when he does that, the system control is returned to the user interface (the AI program), and once again he can choose any of the three options mentioned above.

Choosing option number two will invoke the APMGR program--the program that manages the library of application programs. The integrated system is now under the control of the APMGR program. All the programs contained in this program library can be run through APMGR by simply picking out the program name. When an application program, say PROC, starts running, the

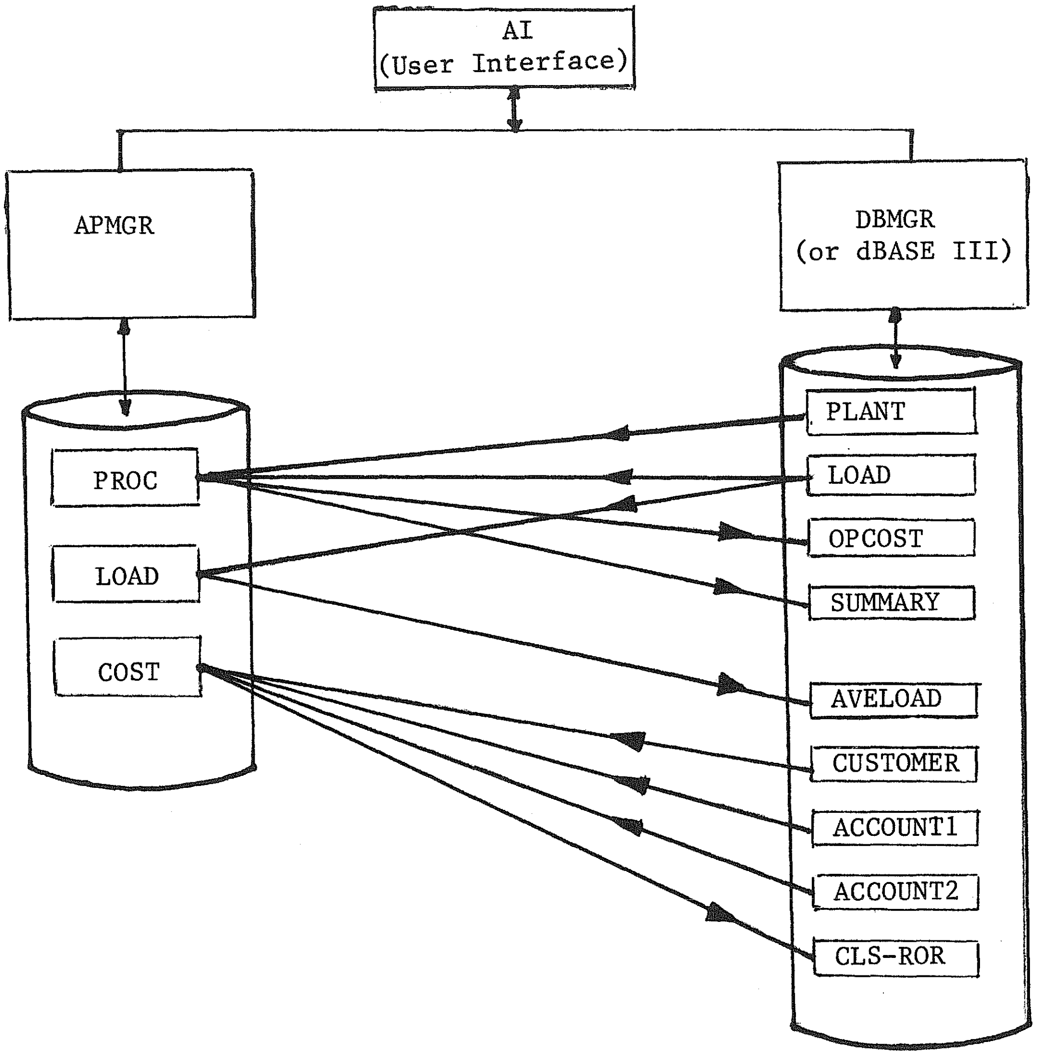

Figure 7. Schematic of the sample integrated system

control of the system is assumed by this particular program. Depending on the design of the program, one or more database files may be used as program input and some other database files may be called upon to store program output. Whichever is the case the user is still working in this integrated environment. When the program finishes execution, the control of the integrated system is returned to the APMGR program. At this point one can either start running another application program, say the LOAD program, or return to the user interface and invoke the DBMS to analyze the program output.

Some observations can be made about this arrangement of applications integration--combined data management and data analysis. First, it provides a work environment in which one can view and/or change program inputs, run individual programs, and analyze program outputs in a coordinated manner, all within the framework of this integrated system. Second, one can switch back and forth among various application programs quickly and efficiently. This is especially useful in situations where a particular analysis requires the use of several independent programs. Furthermore, one can also switch back and forth between the database component and the library of application programs. This feature is very valuable because one can access necessary data and programs in a way that follows his train of thought without many interruptions. This is especially true when one is doing a sensitivity analysis that calls for running a program, changing some data items, and running the program again. Lastly, one can take advantage of some of the more advanced features of the DBMS to present the data contained in the database in a variety of formats and from a variety of viewpoints. For example, with the aid of the DBMS one can almost instantly obtain a listing of those generating units for which the capacity factor exceeds 50 per cent and of which the average running cost is less than 36 mills/kWh.

5. CONCLUSION

A procedure for integrating the capabilities of individual application programs is described and demonstrated in this report. This integration procedure is based on the database approach to data processing system development. Some benefits that this integration procedure offers are highlighted below.

* This procedure should enhance state commissions' problem-solving and decision-making capabilities. It is applicable to all utility sectors, and an example integrated system of three NRRI microcomputer programs was established following this procedure for use in the electric utility sector.

* This integration arrangement--combined data management and data analysis--provides for a work environment in which one can view and/or change program input, run individual programs, and analyze program output in a systematic and consistent manner. This means increased productivity.

* Under this integration arrangement, one can switch back and forth among various application programs quickly and efficiently; this is especially useful in situations where a particular analysis requires the use of several independent programs.

* One can also switch back and forth between the database and application programs, meaning that he can access necessary data and programs in a way that follows his train of thought without many interruptions.

* A high level of consistency in handling data from various sources can be attained because of the data semantics and validation rules imposed upon the database by the DBMS. This could be very significant if the domain of applications integration covers different functional divisions of an organization.

* The integrated system derived from this procedure has an open architecture. New application programs can always be added to the system to handle ad hoc situations, as long as the program interface facility is used for input/output preparations.

* The presence of a DBMS provides one with added flexibility in presenting data in a variety of formats and from a variety of viewpoints.

In summary, the integration procedure described in this report should be of particular use to end users, regulators and utilities alike, who rely on computer programs for problem solving and decision making.

APPENDIX

This appendix contains listings of the application program interface routines. These routines are written based on the internal database file structure described in section 2 of this report. They are provided so that one can custom-tailor his own application programs for use in an integrated system.

```
1000 '
1010 '*** This is a reading routine of the program interface
1020 '*** facility. It is based on the internal database file
1030 '*** structure described in section 2 of the report.
1040 '*** On input, one needs to supply the database file
1050 '*** name (CFILE).
1060 '*** On output,
1070 '***   YEAR% : last 2 digits of the year when file was last updated
1080 '***   MONTH% : the month when file was last updated
1090 '***   DAY%  : ths day when file was last updated
1100 '***   RLENG% : record length
1110 '***   ADDR1% : offset of the data area
1120 '***   NOFLD% : number of data fields in the file
1130 '***   NOREC  : total number of records in the file
1140 '***   CFLD(i,j) : field name (j=1) & field data type (j=2) for the
1150 '***               ith field, 1<=i<=NOFLD%
1160 '***   KSIZE(i,j): field width (j=1) & decimal places (j=2) for the
1170 '***               ith field, 1<=i<=NOFLD%
1180 '***   CDATA(i,j): content of the jth field of the ith record
1190 '***               1<=i<=NOREC, 1<=j<=NOFLD%
1200 '*** Lines 1240 - 1770 deal with the file header and
1210 '*** lines 1780 - 1990 deal with the data area.
1220 '
1230 '
1240 OPEN CFILE AS #1 LEN=32
1250 FIELD 1,1 AS S1$,1 AS S2$,1 AS S3$,1 AS S4$,1 AS S5$,1 AS S6$,1 AS S7$,1 AS S8$,1 AS S9$,1 AS S10$,1 AS S11$,1 AS S12$
1260 CODE%=1
1270 GET 1,1          'get the first 32 bytes of the file header'
1280 '
1290 '*** Parse file header ***'
1300 '
1310 TYPE%=ASC(S1$)  'dBASE III file type,3-->dbf'
1320 YEAR%=ASC(S2$)  'last two digits of the year'
1330 MONTH%=ASC(S3$) 'month'
1340 DAY%=ASC(S4$)   'day'
1350 '
1360 '*** File record length. It is stored in two bytes,'
1370 '*** high byte in S12$, low byte in S11$            '
1380 '
1390 N%=ASC(S12$)
1400 RLENG%=256*(N% MOD 16)+4096*(N%\16)+ASC(S11$)
1410 '
1420 '*** Address of the 1st record (2 bytes),           '
1430 '*** high byte in s10$, low byte in s9$             '
1440 '
1450 N%=ASC(S10$)
1460 ADDR1%=256*(N% MOD 16)+4096*(N%\16)+ASC(S9$)
1470 '
1480 '*** No. of fields defined in the dBASE file '
1490 '
1500 NOFLD%=(ADDR1%-2)/32 - 1
1510 '
1520 '*** No. of records contained in the dBASE file,  '
1530 '*** it is stored in 4 bytes (S5$ - S8$)          '
```

```
1540 '
1550 NOREC=0!
1560 N%=ASC(S8$): IF N%=0 THEN 1580
1570 NOREC=NOREC+16^7*(N%\16)+16^6*(N% MOD 16)
1580 N%=ASC(S7$): IF N%=0 THEN 1600
1590 NOREC=NOREC+16^5*(N%\16)+65536!*(N% MOD 16)
1600 N%=ASC(S6$): IF N%=0 THEN 1620
1610 NOREC=NOREC+4096*(N%\16)+256*(N% MOD 16)
1620 NOREC=NOREC+ASC(S5$)
1630 '
1640 '*** Process field name,field type,field width, and  '
1650 '*** decimal places for each defined data field ;    '
1660 '*** starting at the 2nd 32 bytes block in the file  '
1670 '*** header with every 32-byte pattern repeated for   '
1680 '*** each data field                                 '
1690 '
1700 FIELD 1,10 AS S1$,1 AS NUL1$,1 AS S2$,4 AS NUL2$,1 AS S3$,1 AS S4$
1710 FOR I=1 TO NOFLD%
1720 CODE%=CODE%+1: GET 1,CODE%
1730 CFLD(I,1)=S1$            'field name'
1740 CFLD(I,2)=S2$            'field type: C,N,L,D,M'
1750 KSIZE(I,1)=ASC(S3$)      'field width'
1760 KSIZE(I,2)=ASC(S4$)      'decimal places'
1770 NEXT I
1780 '                                                          '
1790 '*** Beginning of data area. Get every 32 bytes from the I/O  '
1800 '*** buffer; parse and assign them to proper (record,field)   '
1810 '*** pair in the CDATA (record,field) array of string         '
1820 '*** variables                                                '
1830 '
1840 RD%=1:FD%=1:CODE%=NOFLD%+2:P1%=0:P2%=3
1850 GET 1,CODE%:  FIELD 1,32 AS A1$
1860 P1%=P2%+1: P3%=KSIZE(FD%,1): P2%=P2%+P3%
1870 IF P2%>32 THEN 1930
1880 CDATA(RD%,FD%)=CDATA(RD%,FD%)+MID$(A1$,P1%,P3%)
1890 FD%=FD%+1: IF FD%<=NOFLD% THEN 1860
1900 '*** next record ***'
1910 RD%=RD%+1: IF RD%>NOREC THEN 1990
1920 FD%=1: P2%=P2%+1: GOTO 1860
1930 IF (32-P1%)<0 THEN P1%=P1%-32: GOTO 1960
1940 P3%=33-P1%: P1%=1
1950 CDATA(RD%,FD%)=CDATA(RD%,FD%)+RIGHT$(A1$,P3%)
1960 CODE%=CODE%+1: GET 1,CODE%     'get next 32 bytes'
1970 P3%=KSIZE(FD%,1)-LEN(CDATA(RD%,FD%)): P2%=P3%+P1%-1
1980 GOTO 1870
1990 CLOSE #1
```

```
2000 '
2010 '
2020 '*** This is a writing routine of the program interface
2030 '*** facility. It is based on the internal database file
2040 '*** structure described in section 2 of the report.
2050 '*** On input, one needs to supply the following information
2060 '***   (CFILE): database file name
2070 '***   RLENG% : record length
2080 '***   ADDR1% : offset of the data area
2090 '***   NOFLD% : number of data fields in the file
2100 '***   NOREC  : total number of records in the file
2110 '***   CFLD(i,j) : field name (j=1) & field data type (j=2) for the
2120 '***               ith field, 1<=i<=NOFLD%
2130 '***   KSIZE(i,j): field width (j=1) & decimal places (j=2) for the
2140 '***               ith field, 1<=i<=NOFLD%
2150 '***   CDATA(i,j): content of the jth field of the ith record
2160 '***               1<=i<=NOREC, 1<=j<=NOFLD%
2170 '*** Lines 2210 - 2690 deal with the file header and
2180 '*** lines 2700 - 3120 deal with the data area.
2190 '
2200 '
2210 TYPE%=3
2220 OPEN CFILE AS #2 LEN=32: CODE%=1
2230 FIELD 2,1 AS S1$,1 AS S2$,1 AS S3$,1 AS S4$,1 AS S5$,1 AS S6$,1 AS S7$,1 AS S8$,1 AS S9$,1 AS S10$,
     1 AS S11$,1 AS S12$,20 AS NUL$
2240 RSET S1$=CHR$(TYPE%)   'file type'
2250 V$=DATE$
2260 RSET S2$=CHR$(VAL(RIGHT$(V$,2)))   'year'
2270 RSET S3$=CHR$(VAL(LEFT$(V$,2)))    'month'
2280 RSET S4$=CHR$(VAL(MID$(V$,4,2)))   'day'
2290 '*** no. of record to be stored in 4 bytes '
2300 IF NOREC>65535! THEN 2370
2310 L%=NOREC MOD 256   'only needs 2 bytes (s5$,s6$) '
2320 RSET S5$=CHR$(L%)
2330 L%=NOREC\256
2340 RSET S6$=CHR$(L%)
2350 RSET S7$=CHR$(0)
2360 RSET S8$=CHR$(0): GOTO 2480
2370 N%=NOREC/65536!  'needs all 4 bytes   '
2380 L%=NOREC-65536!*N% 'for s5$ & s6$ (<=65535)'
2390 L1%=L% MOD 256
2400 RSET S5$=CHR$(L1%)
2410 L1%=L%\256
2420 RSET S6$=CHR$(L1%)
2430 L%=(NOREC-L%)/65536!  'for s7$ & s8$ (>=65536)'
2440 L1%=L% MOD 256
2450 RSET S7$=CHR$(L1%)
2460 L1%=L%\256
2470 RSET S8$=CHR$(L1%)
2480 L%=ADDR1% MOD 256
2490 RSET S9$=CHR$(L%)     'address of 1st record'
2500 L%=ADDR1%\256
2510 RSET S10$=CHR$(L%)
2520 L%=RLENG% MOD 256
2530 RSET S11$=CHR$(L%)    'record length '
```

```
2540 LZ=RLENG%\256
2550 RSET S12$=CHR$(LZ)
2560 '*** set dummy string NUL$ ***'
2570 NUL$=STRING$(20,0)
2580 PUT 2,CODE%     'write the file header'
2590 '*** write field block ***'
2600 FIELD 2,10 AS S1$,1 AS NUL1$,1 AS S2$,4 AS NUL2$,1 AS S3$,1 AS S4$
2610 FOR I=1 TO NOFLD%
2620 J=LEN(CFLD(I,1)): V1$=CFLD(I,1)+STRING$(10-J,0)
2630 LSET S1$=V1$                    'field name'
2640 LSET S2$=CFLD(I,2)              'field type: C,N,L,D,M'
2650 LSET S3$=CHR$(KSIZE(I,1))  'field width'
2660 LSET S4$=CHR$(KSIZE(I,2))  'decimal places'
2670 LSET NUL1$=CHR$(0): LSET NUL2$=STRING$(4,0)
2680 CODE%=CODE%+1: PUT 2,CODE%
2690 NEXT I
2700 '                                                          '
2710 '*** Beginning of data. Assign each data record (CDATA) to     '
2720 '*** string variable CB; then write to I/O buffer in 32       '
2730 '*** bytes increment.                                         '
2740 '                                                          '
2750 RD%=1: CODE%=CODE%+1: KK=2: FIELD 2,32 AS A1$
2760 IF RD%>NOREC THEN LSET A1$=CHR$(13)+CHR$(0)+CHR$(26): GOTO 3120
2770 CB=CHR$(13)+CHR$(0)+CHR$(32): IF RLENG%<32 THEN 2920
2780 '
2790 '*** record length >= 32
2800 '
2810 FOR I=1 TO NOFLD%: CB=CB+CDATA(RD%,I): NEXT I: I1=1: K=RLENG%+KK
2820 LSET A1$=MID$(CB,I1,32): PUT 2,CODE%: CODE%=CODE%+1
2830 K=K-32: IF K>=32 THEN I1=I1+32: GOTO 2820
2840 IF K>0 THEN 2870
2850 RD%=RD%+1: IF RD%>NOREC THEN LSET A1$=CHR$(26): GOTO 3120
2860 CB=CHR$(32): KK=0: GOTO 2810
2870 V1$=RIGHT$(CB,K):RD%=RD%+1:IF RD%>NOREC THEN 3110
2880 CB=CHR$(32): FOR I=1 TO NOFLD%: CB=CB+CDATA(RD%,I): NEXT I
2890 V1$=V1$+LEFT$(CB,32-K): LSET A1$=V1$: PUT 2,CODE%
2900 CODE%=CODE%+1: K=RLENG%-32+K: I1=1: CB=RIGHT$(CB,K)
2910 IF K>=32 THEN 2820 ELSE 2840
2920 '
2930 '*** record length < 32
2940 '
2950 K=RLENG%+2: KK=32
2960 FOR I=1 TO NOFLD%: CB=CB+CDATA(RD%,I): NEXT I
2970 IF K<32 THEN 3020
2980 LSET A1$=LEFT$(CB,32): PUT 2,CODE%: CODE%=CODE%+1
2990 IF K=32 THEN CB="" ELSE CB=RIGHT$(CB,K-32)
3000 RD%=RD%+1: IF RD%>NOREC THEN V1$=CB: GOTO 3110
3010 CB=CB+CHR$(32): K=K-32+RLENG%: GOTO 2960
3020 V1$=CB: KK=KK-K: K=RLENG%
3030 RD%=RD%+1: IF RD%>NOREC THEN 3100
```

```
3040 CB=CHR$(32): FOR I=1 TO NOFLD%: CB=CB+CDATA(RD%,I): NEXT I
3050 IF KK>K THEN V1$=V1$+CB: KK=KK-K: GOTO 3030
3060 IF KK<K THEN 3080
3070 V1$=V1$+CB:LSET A1$=V1$:PUT 2,CODE%:CODE%=CODE%+1:KK=32:V1$="":GOTO 3030
3080 V1$=V1$+LEFT$(CB,KK): LSET A1$=V1$: PUT 2,CODE%: CODE%=CODE%+1
3090 V1$=RIGHT$(CB,K-KK): KK=32-K+KK: GOTO 3030
3100 IF KK=32 THEN LSET A1$=CHR$(26): GOTO 3120
3110 LSET A1$=V1$+CHR$(26)
3120 PUT 2,CODE%: CLOSE #2
```