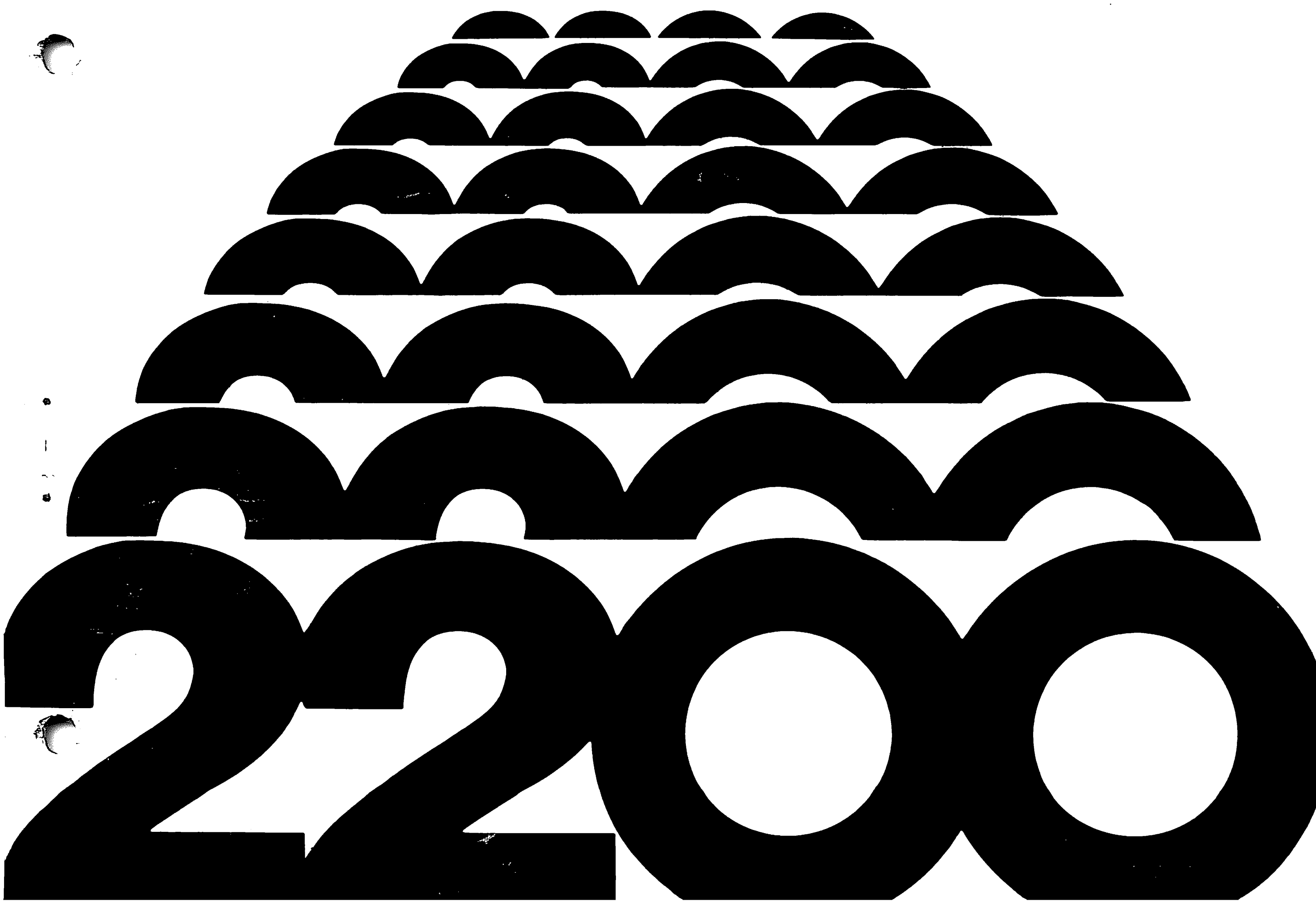
WANG

IDEAS (Inquiry Data Entry Access System) User Manual



IDEAS (Inquiry Data Entry Access System) User Manual

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

TABLE OF CONTENTS

CHAPTER 1
OVERVIEW

1.1 INTRODUCTION

IDEAS (Inquiry Data Entry Access System) is a program development tool designed to facilitate the creation of data entry packages and reports with a minimum amount of programming effort. IDEAS can be used to create and maintain data files, generate sophisticated screen formats, generate menus with or without password protection, solicit and validate operator-entered data, and produce complex reports. Utilization of IDEAS can greatly reduce the programming effort required to produce versatile and comprehensive systems.

IDEAS may be used for different purposes, depending upon the amount of technical knowledge possessed by the user. For the user with some data processing knowledge and a slight familiarity with the concepts of data files, simple applications can be developed and run as produced by IDEAS -- no additional user programming is necessary. For the user with some BASIC programming skills, an understanding of data files, disk I/O and storage, and some knowledge of system design, IDEAS may be used to create skeleton application programs which can be easily modified through the use of the system resident macros. Modification is not difficult because the BASIC code generated by IDEAS is modularized to facilitate use of the macros. IDEAS is designed, however, primarily for the experienced applications programmer or software vendor. The skeleton applications programs generated by IDEAS provides the experienced user with a solid base which can be easily developed into a comprehensive and sophisticated system of application programs through the use of the system resident macros, and additional BASIC programming. This manual primarily addresses the concerns of experienced users and users with some programming experience.

In order to benefit fully from this manual, the following should be read before proceeding: Wang BASIC-2 Disk Reference Manual, Chapters 1 and 2, which discuss accessing a disk drive and the disk catalog; Wang BASIC-2 Language Reference Manual, Chapter 7, which discusses device addresses; the appropriate introductory manual; and the appropriate printer manual.

IDEAS can be used to develop simple applications. However, IDEAS is not designed to generate all or even most applications without programmer intervention. Most applications will require some modification, specifically use of the system resident macros. Complex application systems generally require additional user programming as well. IDEAS should be viewed primarily as a powerful tool which simplifies and expedites programming, and not as a system which produces complete applications programs.

1.2 FUNCTIONAL OVERVIEW

IDEAS consists of two functional parts. The first is the IDEAS System Utilities. The system utilities are used to create data files, screen masks, menus, data entry programs, reports, and START modules. The second part is a set of application files (program and data files) needed to run IDEAS-generated systems. These application files <u>must</u> reside on a disk in the system when an IDEAS-based application is run.

The system utilities are used to create skeleton or simple stand-alone application systems. The user should, before sitting down in front of the terminal, decide what is required of the application to be developed. Fields should be determined, as well as the primary and alternate keys. Since the present version of IDEAS does not have a SORT, special care should be given to the selection of alternate keys. Information can be accessed from the records in different ways using different alternate keys. The user should also decide such things as:

* Are single or multiple volume files to be used?
* What type of primary key is most desirable?
* What types of alternate keys are most desirable?
* What kinds of reports are needed?
* Should the password protection option be used on selected menus? (Or on all menus?)
* How should the data files be organized? Would one large data file be more efficient, or would it be better to have several smaller related data files?

To appreciate the impact of the above decisions, it is strongly recommended that the user read this manual in its entirety prior to beginning his or her first application.

Record lengths should be determined before definition. Each data file may contain up to 128 fields, and each field may be up to 64 bytes long. However, the total record length may not exceed 1008 bytes. The system may allow a record to be defined as more than 1008 bytes; it is assumed that the user is familiar with good data processing procedures, and will plan accordingly. It is recommended that the system utilities be used in the following order:

1. Data File Utilities
2. Application Initialization ("START") Program Generation
3. Application Menu Program Utilities
4. Screen Mask Utilities
5. Data Entry/Inquiry/Update Program Generation
6. Report/Form Printing Utilities

IDEAS is a menu-driven system, which employs a hierarchy of menus. All functions associated with FN'05 to FN'11 remain constant throughout all menus. In other words, it is not necessary to return to the main menu to access other system utilities. IDEAS is screen-oriented in that the user is required to enter information, or select options from screens throughout the system. The primary program selection menu is pictured below. All other IDEAS menus follow this format, and are variations of this menu.

```
IDEAS System Utilities - Primary program selection menu module
==============================================================================
          'FN   Operation
          ---   ---------------------------------------------------

          '00 - System date module
                ------------ Other IDEAS System Utilities ------------
          '05 - Data file utilities
          '06 - Screen mask utilities
          '07 - Report / Form printing utilities
          '08 - Application initialization ("START") program generation
          '09 - Application menu program utilities
          '10 - Data entry / inquiry / update program generation
          '11 - Application module execution
          '31 - Application peripheral device address selection module

  Please touch the special function key corresponding to the desired operation
```

Figure 1-1. IDEAS Primary Program Selection Menu.

All of the fields on the system utilities screens terminate (cursor moves to next field) automatically when full. If the EXEC key is pressed to terminate a screen which is full, the system assumes the EXEC pertains to the next field, which is then terminated. This automatic termination of full fields is incorporated into IDEAS to increase programmer productivity by decreasing the necessary number of keystrokes. It may take some getting used to, but in the long run is much more efficient. Automatic termination of full fields is available through the screen mask utilities for all IDEAS-based systems.

1.2.1 File Management

As part of its run time components, IDEAS provides its own access method, called HIKAM (Hashed Index Keyed Access Method). HIKAM is a comprehensive file management system which combines hashing and indexing access techniques in a way that handles insertions and deletions easily, optimizes data storage and retrieval, minimizes overflow situations, and is significantly faster than indexing. HIKAM performs well in both sequential and random access environments. All disk I/O, and storage is handled automatically by HIKAM.

IDEAS supports both single and multiple volume data files, thus the maximum file size on an IDEAS-developed system is virtually unlimited. A logical file may span up to eight platters on line. The maximum file size is therefore limited only by available memory and disk space. An evaluation of available memory and disk space should be made before deciding whether a multiple or single volume file is desired. Systems vary, and it is not difficult to modify a standard IDEAS-based application system for single or multiple volume files, depending upon the individual requirements of different 2200 configurations.

Each data file must have one primary key; in addition, up to sixteen alternate keys may be associated with each primary data files. Primary keys are stored in the data file itself, while alternate keys are placed in separate files. These alternate key files contain the keys and pointers necessary to manipulate the data in the primary data file for reporting, data entry, inquiry, or modification. Duplicate primary and alternate keys are allowed. Two types of duplicate alternate keys are allowed -- one which optimizes sequential processing, and one which increases efficiency in a random access environment. All key file maintenance is performed automatically by the system.

1.2.2 Data File Utilities

Data file definitions may be created, revised, and documented through the Data File Utilities. The data file definitions contain the attributes of each field in the data file. Each data file must have one primary key associated with it. Up to sixteen alternate key files may be associated with each primary data file. Primary data files and alternate key files are created on disk after the data file definition is completed. To build a file on disk, the user must use the data file utilities to initialize the file.

All fields created with the IDEAS data file utilities are given a unique name and all subsequent references use the field name rather than a variable name. For example, "CUSTNO" would be used in the IDEAS utilities rather than "A$(1)". The BASIC code generated by IDEAS uses variable names. The system resident macros offer subroutines for use with field names and variable names.

Several options are available to the user when creating data file definitions which allow for the most efficient method of disk storage in both random and sequential access modes. Once again, planning is important, since the choices made in these options have a definite effect on file performance, memory requirements, and disk storage allocation. The system automatically computes all relevant information and displays it, allowing the user to select the options which will maximize efficiency in specific applications. When designing a system, the type of processing (random or sequential) most likely to be used in data manipulation should be determined for each primary and alternate key. Then, when the data file utilities are used to create record definitions for primary and alternate key files, the user may intelligently chose the options best suited for his or her individual system.

1.2.3 Application Initialization ("START") Program Generation

"START" modules for IDEAS-based sysetems are created from user inputs in the Application Initialization Program Generation Module. The created START module initializes system addresses and operating parameters, and opens user-specified files. It then loads a user specified program which is typically a "Main Menu" which controls the flow of the application.

1.2.4 Application Menu Program Utility

Menu displays can be quickly created through the Application Menu Program Utilities. Up to thirteen programs or sub-menus can be called per menu. A password security system is available through the menu utilities. Passwords may be assigned to menus during menu creation. Menus with assigned passwords are displayed, but will not allow loading of subsequent programs until the correct password is supplied. All functions within a system requiring security should be accessed through menus with password protection. In this way, unauthorized personnel may not tamper with certain data files, nor print out reports containing confidential information.

1.2.5 Screen Mask Utilities

The Screen Mask Utilities provide the user with an easy-to-use tool for developing the screens necessary for interactive application systems. All the BASIC code necessary to save and reproduce a user-defined screen display format is generated automatically by these utilities. One data file may be specified as a "companion" file for each screen. This companion file is then associated with the creation of the screen display, and certain important field parameters are automatically entered by the system. Other files may be accessed by the screen as well; however, the above mentioned parameters must be entered by the user. The system resident macros are used when accessing information from a data file not specified as a companion file. Information accessed from non-companion files can be caused to appear automatically on the screen, both to reduce data entry effort, and as a means of verification. For example, assume that a payroll system, with employee number as the primary key, is to be developed using IDEAS. The user may choose to have the operator enter an employee number to call certain information, such as the employee's name and address, from another data file. This technique saves the operator time (this information need not be typed in), and allows the operator to check the information to be entered with the information on the screen.

1.2.6 Data Entry/Inquiry/Update Program Generation

Data entry, inquiry, and update programs are generated through the Data Entry/Inquiry/Update Module. These data entry programs allow data manipulation on files created through the Data File Utilities. Programs generated are based on the type of data entry program desired, and the screen and file definitions. Eight different types of data entry programs cam be generated, each with a different set of data entry operations. The BASIC code generated by this utility is highly modularized to facilitate additional user programming and the use of the system resident macros. To create a data entry program, the user need only enter the name of the program to be created, the screen format and data file to be used, and specify the type of data entry program desired.

1.2.7 Report/Form Printing Utilities

The report writer provided by IDEAS allows the user to create report definition files which define the content and format of reports. Each report may access data from up to four data files. The report definition contains specifications for both report format and content. The format specifications include the definition of of the report title and page header, as well as field sequence and spacing between fields. The content specifications allows the designation of new fields for reporting purposes, and includes which data file fields are to be printed, the "sequence" file to be used for the sort order, and whether to set control breaks, and total numeric fields. Since the report definition file is permanently saved, it may be reviewed and revised as reporting needs change.

1.2.8 System Resident Macros

All application programs generated by IDEAS use a subset of the system-resident macros. These macros are a set of 59 powerful subroutine calls designed to minimize programming effort in all phases of IDEAS-developed applications. For example, all file access, key file maintenance, data packing and unpacking, and actual disk operations is performed automatically through the use of one of two system resident macros. For more information on the system resident macros, refer to Appendix A, System Resident Macros.

1.3 START-UP PROCEDURES

IDEAS is delivered on three diskettes: two IDEAS system utilities diskette (one 2270A compatible, the other 2270 compatible), and the IDEAS application utilities diskette. The system utilities diskette contains all the development programs needed to create systems. The system utilities diskette also contains a transfer program, ID-MOVE, to transfer all programs on the diskette to hard disks. ID-MOVE is not 2200T compatible. The IDEAS application utilities diskette contains all the run-time programs needed to run an application and supplementary data file utilities which allow the user to check file status (percent full, etc.), release or protect all records in a file, reconstruct the key files, and convert to or from Wang standard telecommunications file format. ID-MOVE can be used with the application diskette. A backup of both IDEAS diskettes should be made immediately upon receipt.

1.3.1 Hardware Requirements

IDEAS may be used with a 2200T, VP, or MVP system. The minimum memory, disk, and CRT requirements are specified in the chart below:

Category	Requirement
CRT	an 80"X24" CRT screen is required to run IDEAS
Disk	minimally, a dual diskette system is required to run IDEAS. A model 2270A-2 Diskette Drive is strongly recommended.
Memory	2200T -- 32K memory is required to run IDEAS. 2200VP -- 32K memory is required to run IDEAS. 2200MVP -- 32K memory. A 30K partition is required to run the development utilities, a 17.5K global partition is required to run the run-time utilities (individual terminals may be run on less memory).

1.3.2 Storage Considerations

The programmer should determine the disk configuration before starting. If the developed application is to be run from diskette, it is important to store all generated programs on the IDEAS application utilities diskette. (The system utilities diskette does not have room to store any additional files.) Use of a floppy diskette to store an application places an immediate restriction on the size of the application -- the floppy diskette does not provide room to store large systems. If a hard disk is available, the application utilities should be moved to the disk. If possible, the system utilities should also reside on the hard disk as this substantially increases the speed of IDEAS.

An IDEAS-based system makes roughly twice as many entries to a disk catalog index than do non-IDEAS systems. This is because, for most data or program files generated, IDEAS creates a companion file containing IDEAS system control information called a "definition" file. Thus, most IDEAS utilities write two index entries to the catalog for each program or data file created. The following chart lists the types of files created by each utility, the number of sectors contained within files, and the number of entries made to the catalog.

Type of File	Definition File	Program or Data File	# of Index Entries
START	N sector	N sector program	2
Menu	14 sector	4 sector program	2 each menu
Data Entry	None	N sector program	1 each data entry program
Screen	None	18 sector data	1 each screen
Primary Data File	9 sector	*N sector data	2 each data file
Alternate Key File	3 sector	*N sector data	2 each alternate key file
Report	36 sector	4 sector program	2 each report

The definition files are automatically named by IDEAS. The name provided is a lowercase version of the name entered in the utility. Thus, all file names entered must have at least one uppercase letter to distinguish it from the definiton file.

* Number of sectors used displayed during data file creation. See Chapter 3, Section 3.2.5.

The following chart provides a partial list of the contents of each defintion file. For more information on the definition file contents, refer to Appendix B, Data Record Definition Files.

File Type	Contents of Definition File
START	The definition file contains the names of all data files to be opened, and the device addresses to be used by the system.
Data File	The definiton file contains information on the keys, character compression, field names, and the disk address.
Report	The definition file contains information on the report format as specified by the user.
Menu	The definition file contains the program names to the called by the menu, and the assigned PF keys.

The size of the disk catalog in an IDEAS-based system is an issue which the programmer should seriously consider before beginning. If a system is developed with insufficient catalog sectors, problems will crop up months, or even years, after the system is up and running. ID-MOVE is provided to copy the system and applications programs to scratched disks (with sufficient catalogs), and to make backups from one 2270A diskette to another. A general rule-of-thumb formula for determining the size of disk catalogs is:

standard catalog size + application disk catalog size
+ system disk catalog size
+ 2 sectors per major data file + 10 sectors

All 2200 disk catalogs use a hashing alogrithm to store file names and starting sectors. There are several rules pertaining to calalog size which, if followed, optimize the speed of the catalog hashing algorithm. These are:

1. Catalog index sizes which are multiples of three should be avoided; the nature of the hashing algorithm makes catalogs of such sizes extremely inefficient.

2. Even numbers should be avoided; the nature of hashing causes clusters at even sectors.

3. When possible, prime numbers should be used for the catalog size; the hashing algorithm works with remainders, and prime numbers yield non-repetitive remainders. Repetitive remainders tend to produce clustering.

1.3.3 The IDEAS Diskettes

The IDEAS run-time utilities are delivered on two diskettes, one which is 2270 compatible, and one which is 2270A compatible. The 2270A compatible system utility diskette contains, in addition to the development utilities, the run-time utilities (without the supplementary data file utilities). The 2270 compatible system utilities diskette contains only the development utilities. The reason for the difference in the contents of the two diskettes is a result of a difference in the amount of sectors accessible to the diskette drives. The Model 2270 Diskette Drive cannot read files beyond sector 1023, while the Model 2270A may read files up to sector 1232. The development utilities plus the run-time utilities occupy space on the diskette up to sector 1231; the programs up to sector 981 are those needed to run the system utilities, the sectors beyond 981 are the run-time utilities. Since a 2270 diskette drive cannot read beyond sector 1023, all programs occupying sectors 1023 - 1232 are inaccessible on a 2270 drive. On a 2270-based system, these programs must be accessed from the application utilities diskette. Thus the 2270A compatible system utilties diskette should not be used on a Model 2270 Diskette Drive. In particular, ID-MOVE should not be used to transfer files from a 2270A system diskette to a 2270 diskette.

1.3.4 ID-MOVE

The system utilities diskette contains a transfer program, ID-MOVE, to transfer all programs on the diskette to hard disks. ID-MOVE is not 2200T compatible. ID-MOVE can be used with the application diskette. ID-MOVE selects disks 310 and B10 for backup from one diskette to another. If ID-MOVE is to be used to copy the contents of the system and application utilities diskettes to a hard disk, the appropriate output address must be substituted for B10. ID-MOVE should not be used to transfer files from a 2270A system diskette to a 2270 diskette.

Before ID-MOVE can be used, the programmer must substitute the desired output address in line 20 of the program. (A listing of ID-MOVE is provided below.)

```
0010  DIM B$(32)8
0020  SELECT #1 310, #2 B10
0030  FOR 1=0 TO 5
0040  DATA LOAD BA T #1, (I, J)B$()
0050  FOR J=2 TO 32 STEP 2
0060  IF VAL (B$(J)) = 0 THEN 90
0070  PRINT "MOVING FILE . . . . "; HEX(22); STR(B$(J),1,8):
      HEX(2200)
0080  MOVE T #1, B$(J) TO T #2
0090  NEXT J: NEXT I
```

To run ID-MOVE, enter "RUN ID-MOVE". A message will appear on the screen which informs the programmer which file is being moved at any given time.

If IDEAS is to be used on a 2200T, ID-MOVE cannot be used because it is written in BASIC-2, which is not entirely T-compatible. The programmer should read the sections on MOVE and COPY in the Wang BASIC Language Disk Memory Reference Manual before backing up the IDEAS diskettes. Either MOVE or COPY may be used to transfer IDEAS programs to hard disks.

1.4 THE MASTER EXAMPLE

An example referred to as the master example is used throughout this manual to illustrate the discussion of IDEAS. All screens shown in the chapters discussing the development utilities contain information used in developing this example. This section outlines the master example in an attempt to make the discussions throughout the manual clearer to the reader.

Let us assume that The H. Melville Company is a distributor with warehouses in several locations and a large number of customers from a variety of locations. Let us further assume that H. Melville desired a system which would keep track of their products, warehouses, customers, and customer invoices. Thus, the master example contains four primary data files, the warehouse master file (WHRSMSTR), the product master file, (PRODMSTR), the customer master file (CUSTMSTR), and the invoice master file (INVOMSTR). Using IDEAS these files can be created and used in an effective system which meets the needs of The H. Melville Company.

The chapters documenting the development utilities use this example to illustrate all aspects of system creation. However, it is important to realize that the planning stages are perhaps the most vital factors in creating software systems. When trying to envision a software system which fits the needs of a particular company or application, it is important to obtain all information needed to produce a good system specification. IDEAS is a program development tool which substantially decreases programming time; it in no way plans applications. Thus, in order to utilize IDEAS to the fullest potential, first know IDEAS and what is required of the application, and then plan the application.

CHAPTER 2
DAILY SYSTEM INITIALIZATION PROCEDURES

2.1 LOADING IDEAS

To load IDEAS type in "LOAD RUN 'IDEAS'." A screen appears with the valid disk addresses for the IDEAS Utilities displayed.

2.2 THE SYSTEM UTILITIES DISK ADDRESS SELECTION MODULE

The System Utilities Disk Address Selection Module screen is pictured below with values entered from the master example.

```
I.D.E.A.S. System Utilities Disk Address Selection Module          Release 1.0
------------------------------------------------------------------------------

Note:    This module allows the selection of the disk address for the
         drive containing the I.D.E.A.S. System Utility Modules.  The
         disk addresses shown below represent the possible disk addresses
         that may be supported by the I.D.E.A.S. System.  However, not
         all of the possible addresses are available on all system
         configurations.  It is important that you choose only from those
         disk addresses available to you on your particular system.

Allowable disk addresses:                    310   320   330   350   360   370

                                             B10   B20   B30   B50   B60   B70

         D10   D11   D12   D13   D14   D15   D50   D51   D52   D53   D54   D55

         D20   D21   D22   D23   D24   D25   D60   D61   D62   D63   D64   D65

         D30   D31   D32   D33   D34   D35   D70   D71   D72   D73   D74   D75

     [[[  Please enter the I.D.E.A.S. System Utilities' disk address.   D20  ]]]
```

Figure 2-1. The System Utilities Disk Address Selection Screen

If the default address shown in the bottom right corner of the screen is the correct address, touch RETURN. If not, enter the appropriate disk address, from those listed on the screeen, and then touch RETURN. If an error message is encountered, check to see that the disk is loaded at the address entered.

2.3 OVERVIEW OF THE APPLICATION DEVICE SELECTION MODULE

The Application Device Selection Module screen allows the user to set the disk addresses for each of the items listed on the screen. The addresses set in this module will be those used in creating data files, screens, report masks and programs, menus, "START" modules, and data entry programs. These addresses are the default addresses for any "START" modules generated.

2.3.1 The Application Device Selection Module

The Application Device Selection Module screen is pictured below with values entered from the master example. These values may not be available on the user's system. This screen will be referred to throughout the discussion of the Application Device Selection Module.

```
I.D.E.A.S. System Utility - Application Device Selection Module      Release 1.0
================================================================================
Device # 01 / 204  -  Printer address  ( 204  211  212  213  214  215  216 )
Device # 02 / D20  -  Disk address for I.D.E.A.S. System Utilities
Device # 03 / D20  -  Disk address for application screen or report mask files
Device # 04 / D20  -  Disk address for application program files
Device # 05 / D20  -  Disk address for data record definition files
Device # 06 / D20  -  Disk address for application data files
--------------------------------------------------------------------------------
  Note: Application data file disk addresses below are for 2200 VP & MVP only.
--------------------------------------------------------------------------------
Device # 07 / 350 - Disk    Device # 10 / 350 - Disk    Device # 13 / 350 - Disk
Device # 08 / 350 - Disk    Device # 11 / 350 - Disk    Device # 14 / 350 - Disk
Device # 09 / 350 - Disk    Device # 12 / 350 - Disk    Device # 15 / 350 - Disk
--------------------------------------------------------------------------------
Allowable disk addresses: ( Not all addresses may be available on all systems. )
(2200 T, VP, & MVP )  310  320  330  350  360  370
                      B10  B20  B30  B50  B60  B70
(2200 VP & MVP only)  D10  D11  D12  D13  D14  D15  D50  D51  D52  D53  D54  D55
                      D20  D21  D22  D23  D24  D25  D60  D61  D62  D63  D64  D65
                      D30  D31  D32  D33  D34  D35  D70  D71  D72  D73  D74  D75
================================================================================
Touch EXEC to accept as is, or SF Key corresponding to device # to be changed. #
```

Figure 2-2. The Application Device Selection Screen.

If all of the addresses shown on the screen are correct, touch RETURN to continue to the next module. If you wish to change any of the addresses shown, touch the FUNCTION KEY corresponding to the device number address to be changed. (The cursor moves to the first position in the address for the specified device. Enter the desired device address (three characters are required). DO NOT END WITH "RETURN" AFTER KEYING IN 3 CHARACTERS! This will be construed as acceptance of all the addresses, and the next module will be loaded.

2.4 SYSTEM DATE MODULE OPERATING INSTRUCTIONS

When the IDEAS system is loaded, there will be no date input. The cursor appears at the beginning of the six-character date field in the center of the screen. Enter a six-digit date in MMDDYY format.

If the date is valid, the system accepts it, and asks if the date is correct. Touch RETURN to accept the date or EDIT to change the date. If the date is invalid (Ex. 022979 - 1979 is not a leap year and February cannot have 29 days), an error message will appear on the bottom line of the screen. The cursor will return to the date field, and the date must be input again.

If the System Date Module is selected from the primary system menu, RETURN to accept the date, or EDIT to change the date.

NOTE:

If a valid date is not entered in the system date module, fatal errors will occur in program and data file generation modules later.

CHAPTER 3
DATA FILE UTILITIES

3.1 OVERVIEW OF THE DATA FILE UTILITIES

The IDEAS Data File Utilities are used to define, initialize and document all primary data files and alternate key files used by the system. They may also be used to alter parameters of these files and reinitialize them after changes have been made. There are three separate menu selections within the Data File Utilities: New Data File Creation Module, Existing Data File Revision/Reinitialization Module, and Data File Documentation Module.

3.1.1 New Data File Creation - Overview

The file creation module allows the user to specify the name of a new primary data or alternate key file, and instruct the system as to the file's parameters. The user is taken through the module on a screen by screen basis, providing requested information to the system at each screen. When the record has been described, the keys selected, and the file's construction and location decided upon, the user may initialize the file. This initialization procedure performs the basic setup of each data file by creating the file on disk, constructing gross and fine index sectors and pointers to data areas (all of which comprise a "bucket"), and creating a control file on disk which corresponds to the data file. Once a file is initialized, it may be utilized in other IDEAS system modules.

3.1.2 Existing Data File Revision/Reinitialization - Overview

The revision/reinitialization utilities function to help redefine a file which was created with some errors using the creation utilities. Essentially, these procedures display all screens seen in the creation utilities, but with the information pertaining to the file displayed on them. At each screen, the user has the choice of accepting the information as it appears, or of editing the screen by stepping through each entry and changing those which are incorrect.

It must be realized that if an initialized primary or alternate key file is altered using these utilities, all involved files must be reinitialized. Failure to do this will result in improper control information and probable loss of data when subsequently entered. It should also be noted that following the revision/reinitialization procedures properly will destroy all data currently stored in a file. Therefore, these utilities should only be used prior to data entry.

3.1.3 Data File Documentation - Overview

The data file documentation utilities provide necessary information about the contents and structure of data files which is later used in the creation of applications with the data file. For each specified file, three screens appear containing information about aspects of the file. (It is possible with the first screen to revise certain file parameters as well as observe the documentation.) After each of these screens is viewed, the user may optionally print out a descriptive page which combines the most valuable information from the previous screens, including a listing of fields in the data record in both alphabetical and positional order. Documentation is available for both primary data files as well as alternate key files.

3.2 PRIMARY DATA FILE CREATION INSTRUCTIONS

This and the following sections illustrate how to operate the various parts of the IDEAS Data File Utilities. In this section, the primary data file containing customer records (CUSTMSTR) will be used as an example; it is fairly typical of the primary files used in the master example. For background information on the master example, refer to Chapter 1, Section 1.4.

3.2.1 Utility Entry and File Name Specification

The Data File Utilities menu is reached from the main IDEAS system menu by depressing Special Function Key '05 (SF'05). The Data File Utilities menu has three options for data files; the New Data File Creation module is reached from this menu by depressing SF'00.

```
IDEAS Data File Revision/Re-initialization Module                    Release 1.0
================================================================================



Enter file name for data file to be revised or re-initialized

FN'31 will load IDEAS Data File Utilities

                              File Name = CUSTMSTR
```

Figure 3-1. Data File Creation - Name Specification Screen.

The first screen which appears identifies the module and requests the name of the file being created. The user must enter the file name in the eight blocks under which the cursor is located. The file name may be from one to eight characters in length, with the restriction that at least one of the characters be an uppercase letter. (This is necessary because the system creates a companion control file with the same name as the data file, except that all letters in the name are lowercase. The uppercase letter serves to distinguish the data file from its control file.) If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file name already exists, an error message is displayed and the user has the option to give a different file name (EDIT), revise the specified existing file (RETURN), or abort the current procedure (SF'31). If revision is chosen, the Data File Revision/Reinitialization module is loaded; if the procedure is aborted, the system returns to the Data File Utilities Menu.

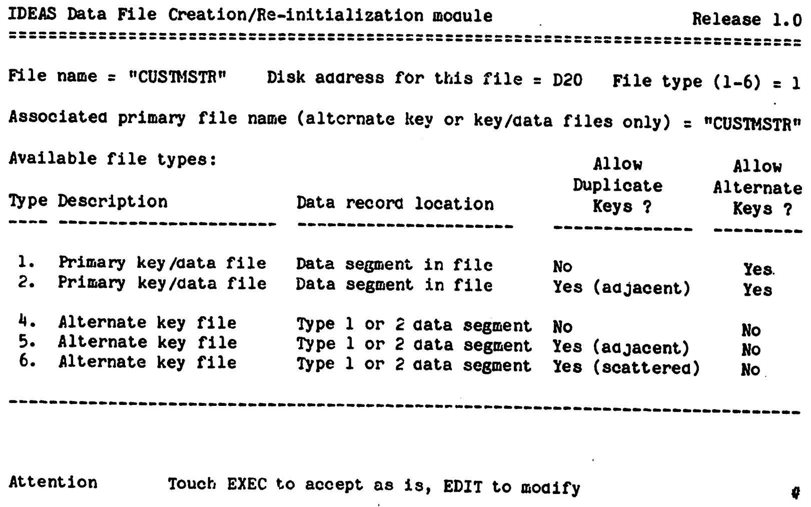
In the example, the user would enter "CUSTMSTR" in the blocks. Since this is an eight-character file name, entry is terminated and the next screen appears.

NOTE:

The use of "IDEAS", "IDEA", "ID-", or similar strings as a file name or part of a file name should be avoided, since most system files incorporate these strings in part for their own names.

3.2.2 Primary Address/File Type Selection Screen

When a primary file name has been specified, the subsequent screen requests two pieces of information; the address of the volume where it will primarily be located and the file type. The primary volume will serve as the sole volume if the file does not span more than one disk; in the case of multi-volume files, the primary volume serves as the base location of the file. The address specified must be one of those chosen as devices #6 through #15 in the disk selection module at the beginning of system operation (refer to Chapter 3, Section 2.), otherwise an error message will be displayed. In the example, the user would enter "D20" and proceed to define the file type.


```
IDEAS Data File Creation/Re-initialization module                 Release 1.0
=============================================================================

File name = "CUSTMSTR"   Disk address for this file = D20   File type (1-6) = 1

Associated primary file name (alternate key or key/data files only) = "CUSTMSTR"

Available file types:                                     Allow        Allow
                                                        Duplicate    Alternate
Type Description              Data record location        Keys ?       Keys ?
---- ----------------------   ----------------------   ------------- ---------

 1.  Primary key/data file    Data segment in file     No               Yes
 2.  Primary key/data file    Data segment in file     Yes (adjacent)   Yes

 4.  Alternate key file       Type 1 or 2 data segment No               No
 5.  Alternate key file       Type 1 or 2 data segment Yes (adjacent)   No
 6.  Alternate key file       Type 1 or 2 data segment Yes (scattered)  No

-----------------------------------------------------------------------------


Attention        Touch EXEC to accept as is, EDIT to modify
```

Figure 3-2. Data File Creation - Address/File Type Screen.

The file type parameter is a single digit keyed to an accompanying table which determines whether the file is primary or alternate, and whether alternate or duplicate keys are allowed. For primary files, there are only two allowable types, "1" and "2". A type 1 primary file does not allow duplicate keys within that file; a type 2 file allows duplicate keys. Both types allow alternate key files to be created which will access primary file information. In the example, type 1 is appropriate and would be keyed in. The system would then respond with the message,

```
Touch EXEC to accept as is, EDIT to modify                    #
```

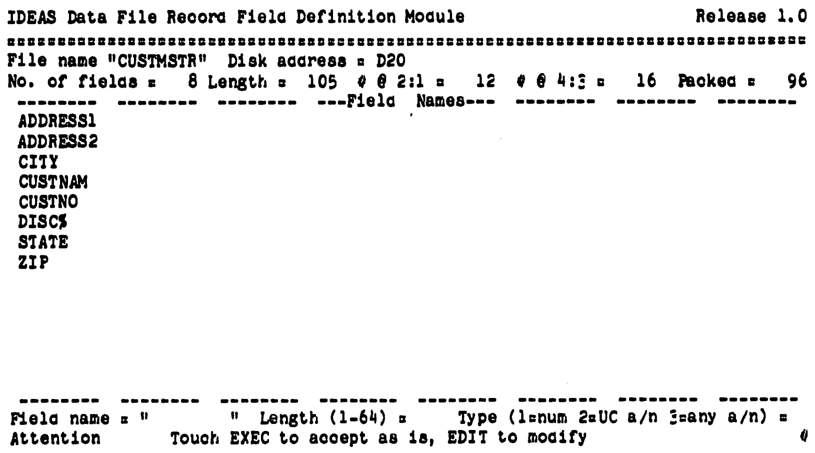
and wait for user entry. Pressing EDIT would position the cursor under the address field and allow a change to be made. The address may be altered by entering a new address, or bypassed by pressing RETURN. The file type may then be altered in the same fashion. Pressing RETURN instead of EDIT causes the parameters to be accepted and the Record Field Definition module to be loaded.

3.2.3 <u>Data Record Field Definition Screen</u>

Once file name, type, and location have been specified, the user must enter the names, lengths, and types of all fields in the data record. Each field name may be from one to eight characters long, and must be unique within the file. They may be specified in any order desired, but will not necessarily be stored in the same order as supplied. Fields are stored in the record in the following order:

First:	All numeric type, in alphabetical order, packed 2:1.
Second:	All uppercase alphabetic type, in alphabetical order, packed 4:3.
Third:	All "any character" type, in alphabetical order, not packed.

As naming of fields progresses, the screen displays all entered fields in alphabetical order <u>without regard to type</u>. This display is for reference purposes only. The screen also keeps a running count of the number of fields, record length, and the number of bytes packed for both numeric and uppercase alphabetic types.



```
IDEAS Data File Record Field Definition Module                    Release 1.0
================================================================================
File name "CUSTMSTR"  Disk address = D20
No. of fields =   8 Length =  105  # @ 2:1 =   12  # @ 4:3 =   16  Packed =   96
 --------  --------  --------  ---Field  Names---  --------  --------  --------
 ADDRESS1
 ADDRESS2
 CITY
 CUSTNAM
 CUSTNO
 DISC$
 STATE
 ZIP



 --------  --------  --------  --------  --------  --------  --------  --------
Field name = "        "  Length (1-64) =     Type (1=num 2=UC a/n 3=any a/n) =
Attention       Touch EXEC to accept as is, EDIT to modify                     #
```

Figure 3-3. Data File Creation - Data Record Definition Screen.

Further restrictions on fields pertain to size of the record, number and size of fields, and size of the key. A maximum record size of 1008 bytes is permitted. While a record longer than 1008 bytes will be accepted at this time, exceeding the maximum length will cause fatal errors at a later point. A maximum of 128 individual fields is allowed, and additional ones will not be accepted. Also, while individual fields are subject to a 64-character length restriction, special attention must be paid to those fields intended to be used as keys. Each key may consist of a maximum three fields with a total length of no more than 58 bytes. Therefore, the key fields must be of such a length that together they will not exceed the maximum key length. For more details on keys, refer to the next subsection.

> NOTE:
>
> If multiple files are to be processed during reporting, the total byte count for all records must not exceed 1008. Additionally, no more than 128 field names total may be processed. For programmer-modified, multiple-file data entry programs, the work required is considerably reduced if the total byte count from all files to be used concurrently is less than 1008.

Each field is entered in the following manner. The system prompts the user to enter the field name in the bottom left corner of the screen. A default field name is displayed and may be used if desired by pressing RETURN. Pressing any other printing key will place that character in the first space of the field name and replace all other positions with blocks. For example, the first default field is "FIELD001". That name may be used by keying RETURN. In the CUSTMSTR example, however, the first field to be entered would be the customer number, "CUSTNO". Therefore, the user would press "C" and the other seven positions would be filled by blocks. When the entire name CUSTNO is entered, RETURN must be keyed to proceed. (If the name is a full eight characters long, e.g., "ADDRESS1," the program proceeds automatically.) Name specified, the system then prompts for field length, an integer from one to 64. In the example, for CUSTNO, a length of 5 would be entered. Finally, the field type must be specified: "1" for numeric, "2" for uppercase alphabetic, and "3" for any character. When the appropriate key (in this case "1") is touched, the system prompts

```
EXEC=New field, EDIT=Revise, FN'09=Delete, FN'20=Exit field edit mode
```

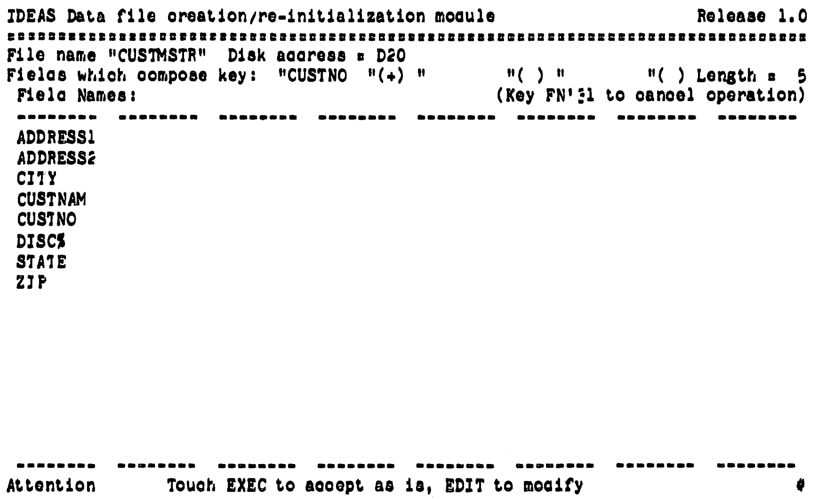
and waits for user response. Keying RETURN allows another new field to be added to the list; it is the normal choice unless the previous field was the final one in the record. The other choices will effect certain optional actions by IDEAS. In the example, fields would continue to be entered by supplying the name, length, and type and pressing RETURN.

Touching EDIT permits a previously entered field to be redefined. When it is keyed, the user must supply the name of the desired field and then may alter either the field length, its type, or both as if the field was being entered for the first time. If an entry is to be deleted, touching SF'09 will permit the user to specify the field name to be removed, verify the action, and delete it from the record.

Pressing SF'20 indicates the end of data field definition and causes the system to verify the choice and load the Data File Performance Option Module. In the example, after all fields have been entered and detailed, SF'20 would be pressed.

3.2.4 Key Field Selection Screen

When the user has defined all fields for a record, the key must be defined. The key is composed of one, two, or three fields within the data record. These fields need not be contiguous, but may total no more than 58 bytes in length. (Again, as in the case of record definition, the user may specify a key that totals more than 58 bytes without system interference. However, this will cause fatal errors later.) The screen displays all fields in alphabetical order, and the cursor is positioned under the blocks corresponding to the name of the first key field. The user would enter the name of that field, and then enter a "+" character if that field is to be used in ascending sort order, or a "-" character if it is to be used in descending sort order. The sort order characters determine in which order keys will be placed in the fine index sectors of each bucket, thereby determining the order in which records will be returned during a sequential scan of the file. If the additional key fields will not be used, pressing RETURN in a blank field will cause the accept/edit prompt to be displayed. This prompt is also displayed after the entry of the sort order for the third key field. Touching RETURN will cause the system to accept the key field selections; pressing EDIT will position the cursor under the first field name and allow alteration of any field name or sort parameter. To edit later entries and leave earlier entries intact, RETURN may be pressed to step through the display until the desired entry is reached.



```
IDEAS Data file creation/re-initialization module                    Release 1.0
================================================================================
File name "CUSTMSTR"  Disk address = D20
Fields which compose key:  "CUSTNO  "(+) "          "( ) "          "( ) Length =  5
 Field Names:                                        (Key FN'31 to cancel operation)
 --------  --------  --------  --------  --------  --------  --------  --------
 ADDRESS1
 ADDRESS2
 CITY
 CUSTNAM
 CUSTNO
 DISC%
 STATE
 ZIP




 --------  --------  --------  --------  --------  --------  --------  --------
Attention       Touch EXEC to accept as is, EDIT to modify                        #
```

Figure 3-4. Data File Definition - Key Field Selection Screen.

In the case of the CUSTMSTR example, the field CUSTNO is the only key field. The name CUSTNO would be typed in, followed by a RETURN (to reach the sort order specification), and finally a "+", indicating a sequential scan of the file will proceed from low customer number to high. RETURN would be keyed again to display the accept/edit prompt, and a second time to exit to the performance option.

3.2.5 Data File Performance Option Selection Screen

After the data record has been defined, but before the file is initialized, the user has the opportunity to "customize" the file. A user must select from several options which will block the records and set up "buckets" according to the primary use of the file. The number of records to be contained in the file, the number of volumes it is to span, and the number of sectors to be allocated on each volume are also specified at this time. This information is then used to create the data definition file which serves as a companion to the main data file. All the described choices are indicated by the user on one screen, which is by nature crowded. The user must carefully note the progression of choices presented by IDEAS. For easier comprehension, these selections will be discussed in detail in the order they will be presented by the system.

```
IDEAS Data File Performance Option Selection Module                  Release 1.0
================================================================================
File Name = "CUSTMSTR" Type 1 (Primary)    No. of records (specified) =      300
Record Length  =  105  # of vols. = 1  Opt#1 = 3  Opt#2 = 4 (actual) =       315
Packed numeric =   12  Vol/Adrs  1/D20 2/    3/    4/    5/    6/    7/    8/
Packed alpha   =   16  Available   133
Packed length  =   96  Required    133
Key length     =    5  Desired     133
--------------------------------------------------------------------------------
Option # 1 ( Record blocking )        Rec / Sec   XtraPad   XtdLen  %Waste   Time
     1. Fastest access time             2 / 1       32       128    25.000    150
     2. Best time/disk compromise       2 / 1       32       128    25.000    150
     3. Most efficient disk use         8 / 3        0        96     0.000    156
Option # 2                     ---RANDOM---  ----------SEQUENTIAL---------- --DISK-
   File structure              ins/del  rtv  1st ]=K   First   Next  Memory  Sectors
   adjustment option             (ms)  (ms)  -(sec)-  -(sec)-  (ms) (bytes)  Needed
     1. Fastest random access    156   156    0.722    0.447   106     132     146
     2. - - - - - - - - - - - -  223   156    0.218    0.168   106      33     135
     3. - - - - - - - - - - - -  306   156    0.162    0.137   106      22     133
     4. - - - - - - - - - - - -  306   156    0.162    0.137   106      22     133
     5. - - - - - - - - - - - -  306   156    0.162    0.137   106      22     133
     6. - - - - - - - - - - - -  306   156    0.162    0.137   106      22     133
     7. Optimum sequential       306   156    0.162    0.137   106      22     133
Attention        Touch EXEC to accept as is & initialize file, EDIT to modify   #
```

Figure 3-5. Data File Creation - File Performance Option Screen.

Initially, the screen displays certain file information in its upper left corner, and then waits at the upper right corner for the user to specify the number of records that will be placed in this file. The number supplied should be the maximum number the file is expected to contain. The system will automatically calculate an expansion factor of at least 5% and no more than 9% and display the calculation below the user-specified number. This calculation is intended not to increase the number of records to be held in the file (although this is possible), but rather to minimize overflow conditions when the file reaches stated capacity. If the bucket to which the record is hashed is full the data overflows and will be stored in the next available bucket. This necessitates extra disk reads and possibly seeks to later obtain the record. Overflows degrade response time and therefore are to be avoided, and exaggeration of file size is a major factor in minimizing them. The system will base all further file calculations on the expanded number of records.

NOTE:

It is strongly recommended that the user put no more records than the number specified into any file. Since related alternate key files may be expanded at a different percentage, based on key size, each may offer a differing number of records when initialized. If there are fewer available records in an alternate key file than in the primary file, it will be impossible to recall any records for which a key could not be placed in the alternate file.

For purposes of the example, a file size of 300 records is adequate for CUSTMSTR. The user would enter 300 and key RETURN. The system calculates a total of 315 records, including a 5% overhead, and displays the result. The cursor would then move to the next entry area.

The user must now specify the number of disk volumes which the file is to span. This may be any integer from one to eight, limited by the number of available volumes on the system. (All specified volumes <u>must</u> be on-line any time the file is used.) A file which spans more than one volume obviously has the potential of being much larger than a single volume file, but there is an additional advantage. It is possible using a multi-volume file to separate index sectors, and therefore keys, from data sectors. If the index sectors should be damaged for any reason, it would be possible to reconstruct that portion of the file from the data stored elsewhere. Of course, a backup of both index sectors and data sectors is strongly urged. The CUSTMSTR file requires only one volume; the user would enter "1" and automatically proceed.

The next portion of the display now provides the user with information on record blocking and asks that a selection be made from among three choices. Option 1 provides a blocking formula which gives the fastest access time. Option 3 blocks for most efficient disk usage, while option 2 attempts to compromise between the two. Sometimes two or even all three options are the same, depending on the structure of the file. Basically, it is a choice between blocking for a minimum number of disk accesses to read or write all potential records in the file, which will probably cause some disk space to be wasted; and blocking for least wasted space, which will cause some records to span two sectors and therefore require extra disk reads to access. For each option, the system will display the number of records and sectors per block (R/S), the number of extra characters added to pad each record so they will fill the allocated sectors, the record length including pads, the percentage of wasted disk space, and the estimated time, in milliseconds, it would take to retrieve a random record. The timing calculation is based on a single-station 2200VP type CPU equipped with a 2260-type hard disk drive.

The choice made here should reflect the intended use of a file. A small file, or a larger one used extensively for on-line inquiry, should probably make use of option 1. Files which are large should probably use option 3, since even a small percentage of wasted space in each block rapidly becomes a problem as it is multiplied. For files which cross onto both sides of the problem, option 2 may be the answer. In the case of the CUSTMSTR file, the consideration is space and option 3 would be selected by pressing "3"; the system would then automatically proceed.

When the blocking option is chosen, the system calculates and displays seven file structure adjustment options. Each option is based on the number of fine index sectors in each "bucket" in the file. Option 1 here is based on a single fine index sector per bucket, and option 7 on the maximum number of fine index sectors for each. The other five choices represent compromises between the two extreme options. Since the ratio of fine index sectors to buckets is not of immediate value to the user in determining file performance, that data is not displayed; rather, system time calculations are shown to enable the user to select the proper option. The use of a single fine index sector in a bucket will provide the fastest random access to the file; using option 7, on the other hand, provides for fastest sequential handling. The display shows calculated times for a random insertion or deletion of a key assuming the file is 75% full, and random retrieval of a record. It also displays sequential timing information for several parameters. The first value is the time calculated to access the first record with a key greater than or equal to that given and build a corresponding MAT MERGE array for sorting. Following that are times for access of the record with the lowest (or highest) key and building of the corresponding merge array; access of the next physical record once the merge array has been built; and the memory size in bytes required for the MAT MERGE array. (The timing information given here is based on the same type of system as those time calculations given in the record blocking option.) Additionally, the number of required disk sectors for each particular option is shown. From this information the user must select the proper option for the data file.

As in the case of record blocking, several of the options may be the same. However, there will always be at least two possible choices. In most cases, one of the compromises is adequate, unless a file is intended for exclusive random or sequential use. Options 4 and 5 are recommended for most files. In the case of CUSTMSTR, option 4 would be chosen and entered by keying "4".

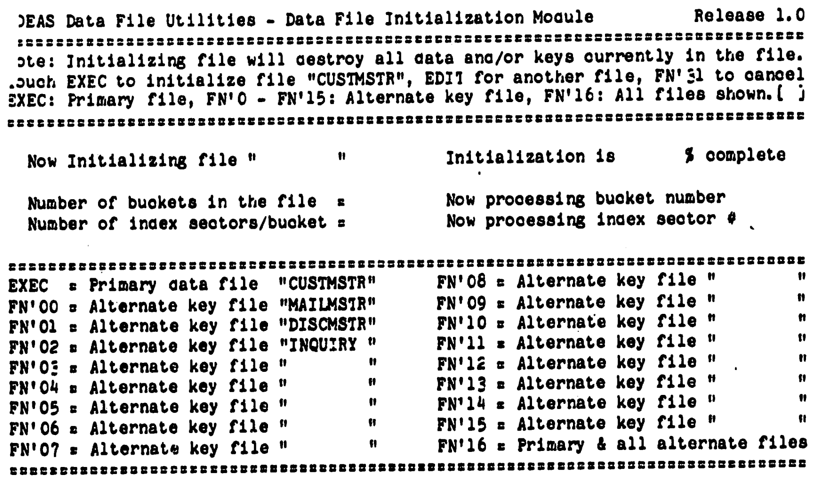
The system automatically proceeds to display the available amount of disk space for each selected volume on which the file is to reside. Beginning with the primary volume specified in the second screen, the address is displayed along with the number of available sectors and the number required by the file. The user must now enter the number of sectors to be allocated to the file on this platter. In the case of multi-volume files, the "required" parameter for the first volume is only sufficient to store keys rather than all data.

For each subsequent volume to be used by the file, the volume address as well as the number of sectors to be allocated may be specified. Providing more space than requested allows for future reinitialization of the file to add more fields to the record. While a file may span up to eight volumes, in the CUSTMSTR example only one volume is required. In this case the required number of sectors, 133, would be entered.

Upon entry of the number of sectors in the final volume, the system displays the accept/edit prompt. Keying EDIT allows respecification of any parameter on this screen. Pressing RETURN accepts the screen as shown and proceeds to the <u>Data File Initialization Module</u>.

3.2.6 Data File Initialization Module

The final step in creation of a data file is initialization. Initialization is the process of actually setting aside the disk space specified for a file and creating empty "buckets" and fine index sectors with corresponding pointers using the information in the data definition file as a guide. A single screen is involved in this module, but several files may be initialized at a time. When the screen first appears, the default file name is the one which was specified when the creation module was first entered. Any associated files (alternate key files for the primary file) which have been defined will also be displayed and may be initialized along with the default file. Any of the associated files whose names appear on the screen may also be initialized singly.



```
)EAS Data File Utilities - Data File Initialization Module          Release 1.0
===============================================================================
ote: Initializing file will destroy all data and/or keys currently in the file.
.ouch EXEC to initialize file "CUSTMSTR", EDIT for another file, FN'31 to cancel
EXEC: Primary file, FN'0 - FN'15: Alternate key file, FN'16: All files shown.[ ]
===============================================================================

  Now Initializing file "          "         Initialization is      % complete

  Number of buckets in the file  =           Now processing bucket number
  Number of index sectors/bucket =           Now processing index sector #

===============================================================================
EXEC  = Primary data file  "CUSTMSTR"     FN'08 = Alternate key file "        "
FN'00 = Alternate key file "MAILMSTR"     FN'09 = Alternate key file "        "
FN'01 = Alternate key file "DISCMSTR"     FN'10 = Alternate key file "        "
FN'02 = Alternate key file "INQUIRY "     FN'11 = Alternate key file "        "
FN'03 = Alternate key file "        "     FN'12 = Alternate key file "        "
FN'04 = Alternate key file "        "     FN'13 = Alternate key file "        "
FN'05 = Alternate key file "        "     FN'14 = Alternate key file "        "
FN'06 = Alternate key file "        "     FN'15 = Alternate key file "        "
FN'07 = Alternate key file "        "     FN'16 = Primary & all alternate files
===============================================================================
```

Figure 3-6. Data File Creation - File Initialization Screen.

It is recommended that all related files (primary data file and associated alternate key files) be initialized at the same time. This saves time and ensures that an alternate key file is not inadvertently left uninitialized. Therefore, it is recommended that all alternate key files be defined before initialization occurs. (Alternate key files cannot be defined before the associated primary file has been defined.)

CAUTION!

Initializing <u>any</u> IDEAS data file will cause the destruction of <u>all information</u> contained in that file!

To avoid the current operation, SF'31 should be pressed. This returns the user to the Data File Utilities Menu. To initialize files, there are two choices: RETURN and EDIT. Pressing RETURN at this time will cause only the default file to be initialized. Pressing EDIT allows the user to then key RETURN to initialize the primary file, SF'00 through SF'15 to initialize individual associated files, or SF'16 to initialize all files. When the choice of files has been made, the system permits the user to proceed by keying RETURN, to change the file selection be keying EDIT, or to abort the procedure by pressing SF'31. Abortion of the file initialization module causes the user to return to the Data File Utilities Menu.

In the case of the CUSTMSTR example, assuming all alternate key files have also been specified, the user would key EDIT to move to the initialization options, then press SF'16 to indicate that all files will be operated on, and finally touch RETURN to begin the process. When initialization of all files is complete (verified by the status display in the screen center), the user is returned to the Data File Utilities Menu.

3.3 ALTERNATE KEY FILE CREATION INSTRUCTIONS

This and the following sections explain how to create an alternate key file associated with an IDEAS primary data file. An alternate key file may only be created after the creation of its corresponding primary data file. For example purposes, the file DISCMSTR, an alternate key file associated with the primary file CUSTMSTR, will be considered. Details on this file and the remainder of the master example can be found in Chapter 1.

Alternate key files are created in much the same way as primary files, from the same menu and with the same screens. However, some differences do exist. These will be noted as the discussion progresses. Frequent references will be made to the previous section on creation of primary files to avoid unnecessary duplication of instructions.

3.3.1 Utility Entry and Alternate Key File Name Specification

Alternate key files are created using the same utilities as primary files. Therefore, the user would begin by entering the utility as described in Section 3.2.1, and then specifying a name in the prescribed manner. In the example, DISCMSTR would be entered in the blocks and the system would proceed to the next screen.

3.3.2 Alternate Key File Address/Type Specification

The next screen is identical to that described in Section 3.2.2, with the exception that when an alternate key file type is specified, the name of the associated primary file must also be provided.

File types for alternate key files are "4", "5", amd "6". The various attributes of each type are detailed below:

- Type 4 - No duplicate keys permitted
- Type 5 - Adjacent duplicate keys allowed
- Type 6 - Scattered duplicate keys allowed

Adjacent duplicate keys are used in the case of a file which is to be accessed randomly; scattered duplicate keys are used only with files to be processed sequentially. When duplicate keys are used, they will all hash to the same home bucket unless some change is made to the algorithm. This will cause the overflow percentage to rise dramatically, thereby increasing access time, especially for sequential files. To alleviate this problem, an alternate key file can be made to "scatter" the duplicate keys within the file by hashing on both the alternate key and the pointer to the record (type 6). This avoidance of overflow aids response time for sequential files. The use of scattered duplicate keys for random access, however, will fail since the system may not find the desired key upon hashing. Instead, it may find one duplicate and will not be able to randomly locate the rest of the duplicate keys. In this case, overflows must be tolerated by using adjacent duplicate keys (type 5). Thus, while the first key found may not be the desired one, issuing a "Get Next Physical" macro instruction will eventually locate the desired key.

In the DISCMSTR example, no duplicate keys will exist, since part of the key will be the unique customer number. Therefore, the user would first enter "D20" as the primary file location, "4" as the file type, and then enter "CUSTMSTR" as the name of the associated primary file. The system will prompt with the accept/edit message, and the user would press RETURN to enter the file description, or EDIT to alter the parameters.

3.3.3 Key Field Specification for Alternate Key Files

The procedure for alternate key file creation bypasses the data record definition screen, since the record has already been described in the primary file. The screen which appears after the file type has been assigned is the key field selection screen, described in Section 3.2.4. Procedures are identical to those described in that section.

In the example, the user would enter "DISC%" as the first key field, followed by a RETURN to bring the cursor to the file type. This field is to be used in descending sort order, so a "-" character would be entered. Then, "CUSTNO" would be entered for the second field, followed by a RETURN and a "+" character would be entered for ascending sort order on this field. Another RETURN, keyed in the empty third field, would call the accept/edit prompt.

3.3.4 Key File Performance Option Selection Screen

With two exceptions, the performance option module operates similarly for both primary and alternate key files. The procedure described in Section 3.2.5 is essentially correct, except that an alternate key file may not span multiple volumes and no record blocking is used. Therefore, if any value other than "1" is entered for the number of volumes, an error message will be displayed. The system will also display "n/a" in the record blocking option area of the screen, and bypass that portion of the module.

The DISCMSTR file will be specified to contain 300 records, as is specified for the CUSTMSTR file. (Alternate key files should always request the same number of records as the primary file, to ensure compatibility.) The user would therefore enter "300" followed by a RETURN. A default value of "1" will appear in the "# of volumes" area; it cannot be changed and RETURN would be entered to proceed. The system then bypasses the record blocking option and displays data in the file structure adjustment area. As in the case of the primary file, the option which best suits the file should be chosen. For DISCMSTR, which will be used sequentially, option 7 would be selected. The volume address appears next, and may be altered to any device address specified as devices #6 through #15 in the device selection module; RETURN then displays available sectors and required sectors. The user would enter "17" to give the required number of sectors and then key RETURN in response to the accept/edit prompt.

3.3.5 Alternate Key File Initialization

Alternate key files may be initialized using the same screen as in Section 3.2.6. However, they should not be initialized separately, but together with their associated primary file and other related alternate key files. Procedures for initialization are given in the referenced section.

3.4 DATA FILE REVISION/REINITIALIZATION INSTRUCTIONS

After a data file has been defined it may be revised using the Existing data file revision/re-initialization module. This module is reached by pressing SF'01 from the Data File Utilities menu. In general, the module proceeds from screen to screen exactly as if the file were being created. However, instead of facing an empty screen, the system displays each screen with file information already filled in. The user may then elect to either accept each screen as displayed or edit the screen. When editing a screen, the user may either be stepped through each entry to verify it, or may be asked specifically which item is to be altered. Therefore, each subsequent subsection will refer to the screen involved and explain which editing method is used and what considerations are involved in changing items there.

If a file that is to be revised has been initialized, it must be reinitialized when revision is complete. In the case of a primary data file, all associated alternate key files must also be reinitialized. If the revision is of a file which has not been initialized, it may be treated in the same way as if it had just been created.

When the module is initially entered, the name of the file to be revised is entered. The name is not subject to revision. When it has been entered, the revision of the file's parameters may begin.

3.4.1 Revision of Address/Type Parameters

The screen which follows name specification is that described in Section 3.2.2. When this screen appears, the prompt

Touch EXEC to accept as is, EDIT to modify

is displayed. The user may bypass any alterations to this screen and proceed to the next by pressing RETURN. If any parameters are to be changed here, though, the user should key EDIT.

```
IDEAS Data File Specification Record Layout Documentation Review     Release 1.0
=================================================================================
Logical File Name  = "CUSTMSTR"                    Number of fields           8
Physical File Name = "CUSTMSTR"                    Record length            105
Primary File Name  = "CUSTMSTR"                    # packed numeric          12
File Type =  1                                     # packed alpha            16
                                                   Packed length             96
Number of volumes = 1                              Last revision date    012980
Volume # 1 Disk Address = D20
Volume # 2 Disk Address =                          Option # 1 (record blocking)= 3
Volume # 3 Disk Address =
Volume # 4 Disk Address =                          Option # 2 (performance)    = 4
Volume # 5 Disk Address =
Volume # 6 Disk Address =                          Associated Alternate Key Files:
Volume # 7 Disk Address =
Volume # 8 Disk Address =                           1. "MAILMSTR"    9. "        "
                                                    2. "DISCMSTR"   10. "        "
Total key length =  5  Key 1     Key 2     Key 3    3. "INQUIRY "   11. "        "
Field name           CUSTNO                         4. "        "   12. "        "
Order                     +                         5. "        "   13. "        "
Position                001                         6. "        "   14. "        "
Length                   05                         7. "        "   15. "        "
Type                      1                         8. "        "   16. "        "
Attention       Touch EXEC to continue to next screen or FN'31 to cancel         #
```

Figure 3-7. Data File Documentation - First Screen.

3.5.2 Editing of Data File Parameters During Documentation

In addition to documentation, some parameters of the file may be edited by pressing EDIT while the first screen is displayed. The two parameters which can be edited are the logical file name (while documenting either primary data files or alternate key files) and the list of related alternate key files (while documenting primary files only). Keying EDIT places the cursor at the beginning of the screen and allows the user to step through some of the displayed parameters.

The logical file name for each file defaults to the physical name of the file as stored on disk. However, if it is desired to logically group several files together, this name may be changed to any user-specified value. In the case of the file CUSTMSTR, the default logical name is CUSTMSTR. However, in this application the user may wish to connect all files to one name, to avoid confusion with another application which may be present. Therefore, the user would enter some name of up to eight characters which would identify the application uniquely.

Using RETURN to step through the screen, the user will eventually arrive at the list of alternate key file names. If the user is documenting a primary data file, any or all alternate key files may be removed from association with the primary file. The user would enter a blank when the cursor is positioned at the desired name, and then press RETURN to effect the change. Pressing RETURN without previously entering a blank just steps through the names without making any changes. For example, if it is desired to remove the alternate key file DISCMSTR from association with CUSTMSTR, the user would first locate the cursor under the name DISCMSTR in the list of alternate key files. Then, the user would press the space bar and then RETURN. The name would be removed from the list and from association with CUSTMSTR.

CHAPTER 4
APPLICATION INITIALIZATION ("START" MODULE) PROGRAM GENERATION

4.1 OVERVIEW OF THE APPLICATION INITIALIZATION PROGRAM GENERATION UTILITY

The Application Initialization Program Generation Utility creates and revises an application initialization module. The created module dimensions all of the necessary common variables to the appropriate size for the specified files, sets certain flags, opens the specified data files, loads the system subroutine module (T and VP), loads the System Date Module, and then loads the specified application module (normally, but not necessarily, a menu module).

4.1.1 The Application Initialization Program Generation Utility

Using the Application Initialization Program Generation Utility is quite simple. The user provides the file name for the start module, the name of the file to be loaded, the device address, and the names of all primary data files to be used by the system. A loading message must be entered, the CANCEL key is specified and the following keys or traps are turned off or on: the skip ahead keys flag, the skip back key flag, the system error message flag, the error message trap, and the function key trap.

The screens from the Application Initialization Program Generation Utility are pictured below with values entered from the master example. These screens will be referred to throughout the discussion of the Application Initialization Program Generation Utility.

```
IDEAS System Utilities - "START" Program Creation/Revision Module     Release 1.0
=================================================================================
Program Name: "STARTDEM"                  Last revision date: 011180 ( JAN 11 80 )
Load Program: "MENU0000"
Load Message: "Herman Melville Co. Data Entry System                              "

Peripheral Device Addresses:                      Available Options:
  Printer ---------------------- # 01 / 204
  IDEAS Utility Disk ---------- # 02 / D20       1.  "CANCEL" FN Key number     31
  Screen/Report Mask Disk ----- # 03 / D20
  Application Program Disk ---- # 04 / D20       2.  Skip ahead keys on    (Y/N) Y
  Data File Description Disk -- # 05 / D20           ( FN'4, FN'11, FN'12 )
  Application Data File Disk -- # 06 / D20
* Application Data File Disk -- # 07 / 350       3.  Skip back keys on     (Y/N) Y
* Application Data File Disk -- # 08 / 350           ( FN'7, FN'13, FN'14 )
* Application Data File Disk -- # 09 / 350
* Application Data File Disk -- # 10 / 350       4.  System error messages (Y/N) Y
* Application Data File Disk -- # 11 / 350
* Application Data File Disk -- # 12 / 350       5.  System error msg trap (Y/N) N
* Application Data File Disk -- # 13 / 350           ( DEFFN '99 )
* Application Data File Disk -- # 14 / 350
* Application Data File Disk -- # 15 / 350       6.  Function key trap     (Y/N) N
* Devices #07-#15 not for use on 2200T CPU           ( DEFFN '98 )
Attention       Touch EXEC to accept as is, EDIT to modify                       #
```

Figure 4-1. First "START" Program Creation/ Revision Screen.

```
IDEAS System Utility - "START" Program Creation/Revision Module      Release 1.0
=================================================================================

Program name: "STARTDEM"
                  [ Application data files to be opened ]
 --------  --------  --------  --------  --------  --------  --------  --------
 CUSTMSTR
 MAILMSTR
 DISCMSTR
 INQUIRY
 PRODMSTR
 WHRSMSTR
 LOCATOR
 INVOMSTR
 SALEFILE
 ALTSALE
 INVOICES
 ALTSALE1
                                                                        --------
                                                                        Number of
                                                                        open data
                                                                        files  12

Attention      EXEC to accept as is, EDIT to modify, FN'0 for previous screen #
```

Figure 4-2. Second "START" Program Creation/ Revision Screen

The first display presented by the Application Initialization ("START") Program Generation Utility requests the name of the data file which will contain the START module. After the START module is defined, it will be saved in this file. The file name must contain at least one uppercase letter. The file name for this start module is STARTDEM. The name of the application program (or menu) module to be loaded must be entered. STARTDEM will load MENU 0000, the main menu of the master example. A loading message (up to 64 characters) to be displayed on the screen is entered at this point. Do not enter "loading" as the system displays "Loading ..." and the specified load message. STARTDEM will display the following message: "Loading Herman Melville Company Data Entry System".

The device address to be used by the system must now be entered. The default values may be accepted by touching return. The "CANCEL" function key number is specified next. The default value is FN'31. The CANCEL key should NOT be FN'0-14, as these keys are used for other functions in the IDEAS-generated application modules. The CANCEL key may be used in a data entry module to return the user to a menu.

The "SKIP AHEAD" key flag must be turned on or off. The skip ahead keys are:

FN'04	Go to the end of the screen.
FN'11	Skip 5 fields.
FN'12	Skip 1 field.

These are set to default value of "N" (or "off") so that an operator in a data entry module may not skip a required field. If the user desires, this default value may be changed to yes. Turn the "SKIP BACK" keys on or off. The skip back keys are:

FN'07	Skip back to the first field.
FN'12	Skip 1 field.
FN'13	Skip 5 fields.

The system error message flag must be turned on or off. If this is off, no error message will be displayed. It is recommended that the system error message flag is turned on; however, it may be turned off and replaced with the user's own error message routines. The error message trap must be turned on or off. This option should be exercised only by the sophisticated user. When the error message flag is turned on, IDEAS assumes that there is a subroutine labeled "DEFFN'99" in each application program. Whenever a system error is encountered, the program will branch to this routine. If DEFFN'99 does not exist, a fatal error will result.

The function key trap must be turned on or off. This option should also be exercised only by the sophisticated user. When the function key trap is turned on, IDEAS assumes that each application program contains a subroutine labeled "DEFFN'98". Whenever a function key is touched, the program will branch to this routine. If DEFFN'98 does not exist, a fatal error occurs.

When all of the above data has been entered correctly, touch RETURN to proceed to the next screen, or touch EDIT to modify any of the fields above. (If EDIT is touched, use the RETURN key to position the cursor at the desired field.) The second Application Initialization Program Generation Utility screen appears upon copletion of the above data. All the names of all PRIMARY data files to be used in the application must be entered in this screen. Any alternate key files which pertain to the primary files will be automatically determined. In the case of a multi-volume file, enter the primary file name once. It will be repeated by the system for as many volumes as have been specified.

NOTE:

A maximum of 124 files may be open at one time.

After all primary file names have been entered, RETURN to enter all information, or EDIT to modify.

CHAPTER 5
MENU PROGRAM UTILITY

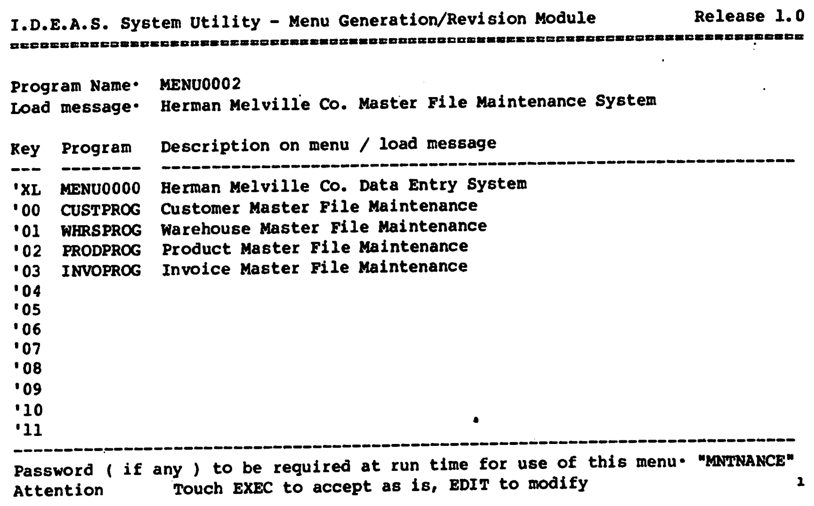
5.1 OVERVIEW OF THE MENU PROGRAM UTILITY

The Menu Program Utilities create, revise, and document menus. Up to thirteen sub-menus or programs can be called per menu. A password security system is available through the Menu Program Utilities; passwords may be assigned to menus during menu creation. Menus with assigned passwords are displayed, but will not allow loading of subsequent programs until the correct password is supplied. All functions within a system requiring security should be accessed through menus with password protection. In this way, unauthorized personnel are denied access to sensitive information.

5.1.1 The Menu Program Utility

Using the Menu Program Utility is very easy. The user enters the name of the menu to be created, the load message to be displayed as the menu is loaded, and the program file names of all menu entries, as well as menu entries.

The menu utility screen is pictured below, with values taken from the master example. A copy of the menu documentation generated for this menu is pictured later in this chapter. Both will be referred to throughout the discussion of te Menu Program Utilities.



```
I.D.E.A.S. System Utility - Menu Generation/Revision Module        Release 1.0
==============================================================================

Program Name·  MENU0002
Load message·  Herman Melville Co. Master File Maintenance System

Key  Program   Description on menu / load message
---  --------  ---------------------------------------------------------------
'XL  MENU0000  Herman Melville Co. Data Entry System
'00  CUSTPROG  Customer Master File Maintenance
'01  WHRSPROG  Warehouse Master File Maintenance
'02  PRODPROG  Product Master File Maintenance
'03  INVOPROG  Invoice Master File Maintenance
'04
'05
'06
'07
'08
'09
'10
'11
------------------------------------------------------------------------------
Password ( if any ) to be required at run time for use of this menu· "MNTNANCE"
Attention       Touch EXEC to accept as is, EDIT to modify                   1
```

Figure 5-1. The Menu Program Utility Screen.

The menu used in this example is taken from the master example. MENU0002 contains entries for all data file maintenance programs in the Herman Melville Data Entry System. Since the data file maintenance functions contained in MENU0002 should be performed by authorized personnel only, MENU0002 is password protected.

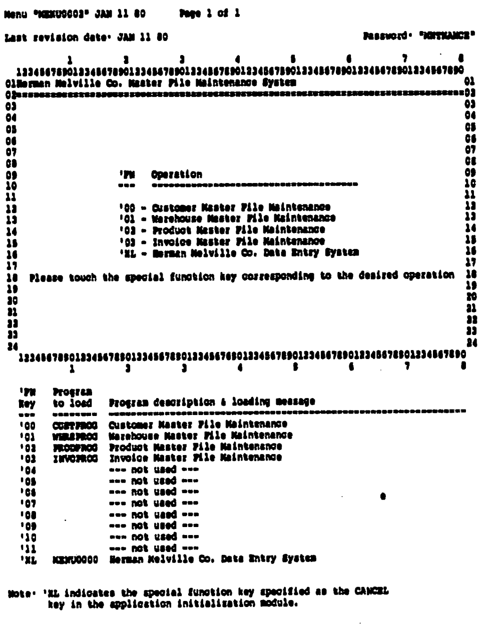
The user is first requested to enter the name of the data file which will contain the menu definition. After the menu is defined, it is saved in this file. After the file name for the menu to be created is entered, the load message must be entered. "Herman Melville Co. Master File Maintenance System" will be displayed as MENU002 is loaded.

The file names of the programs to be called from the menu must be entered next. A load message must accompany each program file name. Note that the function key numbers are placed next to the file names in the created menu. The function key number next to each entered file name becomes that by which the program is accessed. FN'XL is the CANCEL key. The CANCEL function key is assigned during Application Initialization Program Generation. For more information, refer to <u>Chapter 4, Section 4.1.1 The Application Program Generation Utility</u>. All unused fields must be stepped through by using the RETURN key to maneuver the cursor.

Once all file names and load messages are entered, touch RETURN to position the cursor at the password section at the bottom of the screen. Enter the desired password if the menu is to be password protected. If a password is not desired, touch RETURN.

Once all fields have been entered, touch RETURN to accept the menu. The menu documentation is printed automatically at this point. If documentation of menus is desired at a later date, select the menu documentation module (from the Menu Utility menu), enter the name of the menu to be documented and RETURN. The documentation will be printed.

The documentation for MENU0002 is provided below.



```
Menu "MENU0002" JAN 11 80      Page 1 of 1

Last revision date: JAN 11 80                                      Password: "MSTMANCE"

          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
01Herman Melville Co. Master File Maintenance System                              01
02================================================================================02
03                                                                                03
04                                                                                04
05                                                                                05
06                                                                                06
07                                                                                07
08                                                                                08
09                     'FN   Operation                                            09
10                     ---   -------------------------------------                10
11                                                                                11
12                     '00 - Customer Master File Maintenance                     12
13                     '01 - Warehouse Master File Maintenance                    13
14                     '02 - Product Master File Maintenance                      14
15                     '03 - Invoice Master File Maintenance                      15
16                     'XL - Herman Melville Co. Data Entry System                16
17                                                                                17
18  Please touch the special function key corresponding to the desired operation  18
19                                                                                19
20                                                                                20
21                                                                                21
22                                                                                22
23                                                                                23
24                                                                                24
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
          1         2         3         4         5         6         7         8

 'FN   Program
 Key   to load    Program description & loading message
 ---   --------   ------------------------------------------------------------
 '00   CUSTPROG   Customer Master File Maintenance
 '01   WHSEPROG   Warehouse Master File Maintenance
 '02   PRODPROG   Product Master File Maintenance
 '03   INVCPROG   Invoice Master File Maintenance
 '04              --- not used ---
 '05              --- not used ---
 '06              --- not used ---
 '07              --- not used ---
 '08              --- not used ---
 '09              --- not used ---
 '10              --- not used ---
 '11              --- not used ---
 'XL   MENU0000   Herman Melville Co. Data Entry System

Note: 'XL indicates the special function key specified as the CANCEL
      key in the application initialization module.
```

Figure 5-2. Menu Program Utility Documentation on MENU0002.

CHAPTER 6
SCREEN MASK UTILITIES

6.1 OVERVIEW OF THE SCREEN MASK UTILITIES

The IDEAS Screen Mask Utilities are used to define a formatted screen display (a "mask") which will solicit operator input for the purpose of creating or updating a data file. A screen mask may contain operator instructions, headings, and data field descriptions - any information, in fact, which will help the operator understand what is required and enter the correct data. The screen mask (or masks) created with these utilities will be displayed by an IDEAS-generated data entry program (see Chapter 7) to solicit operator entered data which will be posted to an IDEAS-created data file (see Chapter 4).

The set of Screen Mask Utilities consists of three modules: a screen mask definition module, a screen mask revision module, and a screen mask documentation module. Detailed operating instructions for each module are listed below. For screen mask definition, the program displays a blank screen into which the user simply types any text which is to appear in the display, positioning it at any desired location on the screen. Next, the field attributes must be defined for each data entry field in the display. Field attribute options include such items as field length and type, whether the field is required or optional, whether it is to be left or right justified and zero or space filled, etc. (The complete set of field attribute options is listed and explained in Section 6.2). When all desired text has been entered and all fields have been defined, the screen mask is saved on disk for subsequent use by a data entry program. For each screen mask defined by the user, IDEAS creates a screen mask definition file containing all information required to reconstruct the display and define all data fields. This definition file is used by an IDEAS-generated data entry program to recall the display for data entry purposes.

The screen mask revision module permits the user to recall and modify an existing screen mask. Note, however, that IDEAS does not provide, as a standard feature, the capability to make a copy of an existing screen mask definition which can then be modified to create a new mask.

The screen mask documentation module permits the user to print a copy of the screen mask just created along with a listing of field attributes for each data field in the display.

6.2 SCREEN MASK CREATION

To invoke the Screen Mask Utilities, depress FN from the IDEAS Master Menu. Then use FN'0 to run the screen mask creation module.

6.2.1 Screen Mask Definition File Name

The first display presented by screen mask creation requests the name of the data file which will contain the screen mask definition. After the mask is defined, it will be saved in this file; subsequently, it is this file name which must be supplied to the Data Entry Program Generation Utility in order to associate this screen mask with a data entry program. The file name may be 1-8 characters in length.

The mask which will be created as part of our example is a rather simple inquiry screen which allows the program user to inspect a customer file on the basis of customer discounts. (The example screen mask itself is shown in Figure 6-1). The name we assign to this screen mask is "INQSCRN" and this is the name which should be entered opposite File Name. Since the name is not a full eight characters long, the user must key RETURN to enter the data.

6.2.2 Associated Data File Name

Next, the user is requested to enter the name of a data file which will be associated with this screen mask when the data entry. If a data file name is entered at this time, IDEAS will extract field parameters from the definition file associated with this data file and provide them as default values for corresponding fields in the screen mask. The user is thereby spared the task of redefining fields in the display which have already been defined for the data file. Although it is not strictly necessary to specify a companion data file at this time, good design practice dictates that data file formats be defined prior to screen formats. It is therefore strongly recommended that the data file to be associated with this screen be defined first (see Chapter 5) and specified as the companion file when the screen is created.

NOTE:

If a comparison data file is specified for a screen mask, the data file (and, more importantly, its associated definition file) must already have been created through the Data File Definition Utilities. Otherwise, IDEAS will signal an error when it attempts to open the definition file in order to obtain field parameter information for the screen mask.

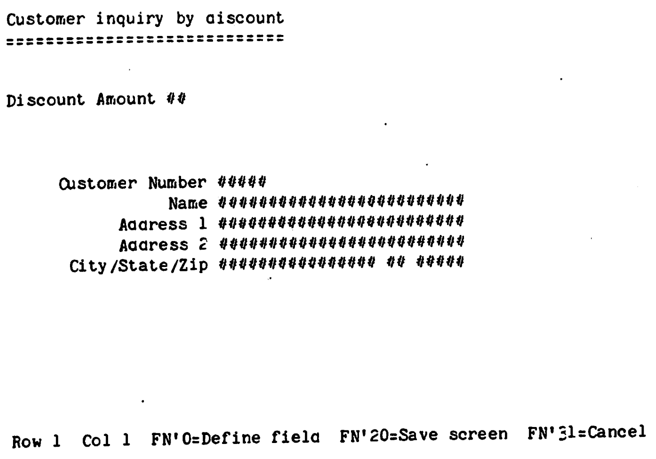
IDEAS itself permits only one data file to be associated with each screen; it is, however, possible to combine fields from several files on one screen with some customization of the data entry module. This procedure requires special attention to parameter definition for the fields taken from a second and subsequent data files. Refer to Section 6.6 for a complete discussion of this problem.

For the present example, the comparison data file is named "INQUIRY". (If this file has not already been created as part of two sample exercise, leave this field blank.)

6.2.3 Screen Mask Creation

The utility now displays the Mask Editor screen. If a new screen mask is to be defined, this screen is blank. The programmer may enter any text which is to appear in the formatted display, including headings, operator instructions, and field descriptions. All edit keys are operational.

Text can be located at any position on the screen simply by positioning the cursor to the desired location and typing the desired text. The current cursor position is always shown at the bottom left corner of the screen.



```
Customer inquiry by aiscount
============================


Discount Amount ##



    Customer Number #####
               Name #########################
          Aaaress 1 #########################
          Aaaress 2 #########################
     City/State/Zip ################ ## #####







Row 1  Col 1  FN'0=Define fiela  FN'20=Save screen  FN'31=Cancel
```

Figure 6-1. Inquiry Screen Mask Created with Mask Editor

6.2.4 Data Field Definition

Each field in the display must be defined by specifying appropriate field parameters for that field in the Field Parameter Selection Screen. To define a new field, position the cursor to the screen location which will be the starting position of the field and depress FN'0 to exit from the Mask Editor and invoke the Field Parameter Selection screen.

To modify the field parameters for an existing field, position the cursor any place within the field and depress FN'0.

The Field Parameter Selection screen is now displayed. Figure 6-2 shows a parameter selection screen with field parameters defined for the first field in the example screen mask in Figure 6-1 ("Discount Amount").

```
I.D.E.A.S. System Utility - Create / Revise Screen "INQSCRN "        Release 1.0
================================================================================
'FN  Field parameter ------------------------------    ---------------------
'00  Field name ? ........................"DISC$   "    Valid Character List
'01  Row on screen ? .............................. 5    1 Digits only
'02  Column on screen ? .......................... 17    2 Digits & decimal pt
'03  Position in Record ? ......................... 6    3 Digits & signs
'04  Default field ? .............. No default field    4 Digits, signs, & dec
'05  Field length ? ............................... 2    5 Upper case letters
'06  Valid characters ? ................ Digits only    6 UC alpha & digits
'07  Allow keyboard entry ? .................... Yes    7 UC, digits, & punct
'08  Allow display ? ........................... Yes    8 Any character
'09  Required or optional ? ......... Required field    9 FN Keys, EDIT & EXEC
'10  Full if present ? .......... Need not be filled    ---------------------
'11  Left or right justified ? ..... Right justified
'12  Zero or space fill ? ............. Space filled
'13  Number of decimal places ? .................. 0
'14  Termination Full/EXEC ? ... Terminate when full
'15  Save field parameters
---------------------------------------------------
'25  Delete current field  '31 Cancel to mask editor
'26  Insert current field  EDIT Display screen mask

Touch the FUNCTION KEY corresponding to the desired operation.
```

Figure 6-2. Field Parameter Screen For First Field ("Discount Amount") in Inquiry Screen Mask.

The available field parameter options are discussed in detail below:

FN'00 Field Name - This name identifies the display field for purposes of transporting data between the display and an associated data file. The name may any combination of up to eight characters (embedded spaces are allowed). The utility automatically supplies a default name (starting with "Fld #001") which may be changed by the programmer. If a companion data file was specified for this screen mask and a field name is entered which corresponds to the name of a field in the data file, certain parameters for that field are taken from the companion file and displayed as default values here. The parameters displayed as defaults include field length, position in record, and most general valid character set. Note that in this case, the length and position <u>cannot</u> be changed. The valid character set may be changed, but only to a character set which is <u>less</u> general than the defualt (i.e., new character type must be less than or equal to the default type: see FN'06 - valid characters).

FN'01 Row on Screen - This is the default value from the current cursor position on the screen for a new field, or from the starting position on the screen for a field being edited. This value may be changed.

FN'02 Column on Screen - See FN'01

FN'03 Position in Record - The starting position of this field within the record buffer. If this field corresponds to one from a companion data file, the positon is defaulted from the data file and cannot be changed. If there is no corresponding companion data file field, this parameter defaults to the next available position in the record buffer. Note that great care must be exercised in determining field position when fields from several files are to be used for this screen; refer to Section 6.6 for a duscussion of this problem.

FN'04 Default Field - This name of a field which is to supply a default value for the current field. Another field previously defined on this screen or the current field itself may be specified as the default field. If another field is specified, the value entered for that field is moved at run time to the current field. If the current field itself is specified as the default, the previous contents of the field are redisplayed (i.e., the default is taken from the current field in the previous record.)

FN'05 Field Length - The length of the field in bytes; must be greater than zero and less tha 65. For numeric fields, the length should not exceed 13 (13 is the maximum number of digits for a numeric value). Numeric values containing more than 13 digits will be truncated.

FN'06 Valid Characters - The character set from which characters will be accepted as legal entries for this field. There are nine possible character sets from which to choose, identified by the numbers 1-9:

1 - Digits only
2 - Digits and decimal point
3 - Digits and signs
4 - Digits, signs, and decimal point
5 - Uppercase letters only
6 - Uppercase letters and digits
7 - Uppercase letters, digits, and punctuation characters
8 - Any character
9 - FN keys, EDIT, and EXEC only

FN'07 Allow Keyboard Entry? - A value of "Y" for this parameter means that the operator may enter values for this field from the keyboard (and thereby, presumably, modify the contents of this field in the data file). A value of "N" indicates that the field is non-modifiable by the operator.

FN'08 Allow Display? - A value of "Y" means that the field will be displayed on the screen at run time; "N" means that it will not be displayed. Nondisplayed, or "hidden," fields are useful for containing information which must be posted to the data file but which is not of interest to the operator. For example, fields from several data files which provide default values for fields on the screen should be hidden since their presence in the display would only serve to confuse the operator. See Section 9.5 for a further discussion of this issue.

NOTE:

If a nondisplayed field is to be required, specify required (FN'09) before specifying no display.

FN'09 Required or Optional? - A value (other than all blanks) may be required or not for this field. If a default field is specified, the default value can satisfy the requirement without operator input.

FN'10 Full if Present? - For both required and non-required fields the programmer may specify that the field must either be filled completely (exluding blanks) or left all blank. For example, Social Security Number might be an optional field, but if anything is entered, the complete 9-digit number must be entered.

FN'11 Left or Right Justified - A field may be either left or right justified.

FN'12 Zero or Space Filled - A numeric field which is right justified, may be padded on the left with either zeros or spaces if completely filled.

FN'13 Number of Decimal Places - A number of decimal places may be specified for a numeric field. At run time, the number entered by the operator is the rounded to the specified number of places if it contains a decimal. If a decimal point is not entered, the implied decimal point is assumed to be N places to the left of the last digit entered, where N is the number of specified decimal places.

The exception to the above is when there are 0 decimal places specified. Then the number entered is not adjusted.

If the field is a designation field for a computed result, 0 decimal places causes the number to be rounded to the nearest integer value.

FN'14	Termination FULL/EXEC - A field may be terminated automatically when full, or EXEC may be required.
FN'15	Save Field Parameters - Depressing this FN key will cause the current field parameters to be entered into the field parameter array and control will be returned to the screen mask editor. The fields are NOT yet saved on disk, however; that does not occur until the screen is saved (FN'20 from the Screen Mask Editor display).
FN'25	Delete Current Field - Depressing this key will give the user the option to delete the current field from the screen; care must be taken in that any default field values concerning this field or subsequent fields must be manually adjusted by the user. (See Section 6.5 for a discussion of this problem.) (The programs provide an opportunity to recover if this key is touched inadvertantly.)
FN'26	Insert Current Field - Depressing this key gives the user the option to insert a new field between existing fields. Care must be taken to manually adjust any subsequent default fields. (See Section 6.5 for a discussion of this problem.)
FN'31	Cancel To Mask Editor - Depressing this key cancels the current field definition operation and returns to the screen mask editor.
EDIT	Depressing this key displays the screen mask without losing any of the current parameters that may already have been entered for a new field. The current field is displayed using diamonds instead of boxes for ease of identification (if length is non-zero). Touching any key causes a return to the field definition module.

6.2.5 Saving the Screen Mask on Disk

When the screen mask is completed and all fields have been defined, use FN'20 from the Mask Editor to save the defined screen. A screen mask definition file will be created and given the name specified for this screen.

Alternatively, FN'31 can be used to cancel the screen definition/ revision module without saving any of the screen information just entered.

Before the screen is actually saved, the utility displays a message requesting the run-time display option to be used when the defined screen is displayed by a data entry program. The available options are:

0 - Display each field at entry only. In this case, each field in the screen is displayed only after the preceding field has been entered.

1 - Display all fields when screen is loaded. In this case, the entire screen, including all fields, is displayed when the screen is called up by the data entry module at run time.

2 - Resume screen edit. The screen is not saved, the utility returns to the Mask Editor, and the user may continue editing the screen mask.

After the screen is saved, the Screen Mask Documentation Module is automatically loaded.

6.3 SCREEN MASK DOCUMENTATION

The Screen Mask Documentation Module can be used to produce a hard copy of the defined screen mask along with a listing of field parameters for all fields defined on that screen. There are two options:

EXEC (RETURN)	- Document the screen mask. (A printer must be available and selected.)
FN'31	- Return to Screen Mask Definition Utility Menu <u>without</u> producing screen documentation.

6.4 SCREEN MASK REVISION

The Screen Mask Revision Module (FN' from the master menu) can be used to revise an existing screen mask. Enter the name of the screen to be revised and its companion data file, if any. The screen mask is then loaded from disk and displayed. All editing and field parameter definition options operate as described in Section 6.2.

6.5 SPECIAL CONSIDERATIONS FOR INSERTING AND DELETING FIELDS WHEN DEFAULT FIELDS ARE USED

When one or more default fields are specified on a screen, inserting or deleting fields on that screen can cause problems which require special attention from the programmer. The problem arises because default fields are identified internally by field number rather than field name. When a new field is inserted or an existing field is deleted, the field numbers of all fields after the affected field are changed. In this case, IDEAS automatically adjusts each default field reference to point to the field whose <u>field number</u> is the same as the field number of the originally specified default field. If the default field has acquired a new field number as the result of an insertion or deletion, it will <u>no longer appear</u> as the default field for the current field. The programmer can correct this problem only by manually changing default field references to the correct field names for each field which references a default.

An example may help to make the problem, and its solution, somewhat clearer. Consider Figures 6.4 and 6.5. Figure 6-4 is a portion of the screen documentation for a screen with four fields, named Fld #001 through Fld #004. Notice that Fld #003 specifies Fld #002, the second field in the record, as a default field. Fld #004 specifies itself as a default. Figure 6-5 shows the documentation for the same screen <u>after</u> a new field, Fld #005, has been inserted at the beginning of the record. Since Fld #005 is now the first field in the record, the field numbers of all other fields are increased by one (that is, Fld #001 now becomes the <u>second</u> field in the record, etc.)

```
Screen "SCREEN1" NOV 22 33               Page 2 of 2

Field parameters:               (field number order)
Name      No  R  C  L  Pos  Valid Characters     Req Ful J F Term Kbd Dis D Dup Field
-------- --- -- -- -- ---- -------------------- --- --- - - ---- --- --- - ---------
Fld #001 001 05 21 05 0021 U/C alpha & numerics  No  No L - Full Yes Yes -
Fld #002 002 06 21 05 0006 U/C alpha & numerics  No  No L - Full Yes Yes -
Fld #003 003 07 21 05 0011 U/C alpha & numerics  No  No L - Full Yes Yes -  Fld #002
Fld #004 004 08 21 05 0016 U/C alpha & numerics  No  No L - Full Yes Yes - *Fld #004
```

Figure 6-4. Screen Field Documentation Showing Default Fields Before Field Insertion.

```
Screen "SCREEN1" NOV 22 33               Page 2 of 2

Field parameters:               (field number order)
Name      No  R  C  L  Pos  Valid Characters     Req Ful J F Term Kbd Dis D Dup Field
-------- --- -- -- -- ---- -------------------- --- --- - - ---- --- --- - ---------
Fld #005 001 09 21 05 0026 U/C alpha & numerics  No  No L - Full Yes Yes -
Fld #001 002 05 21 05 0021 U/C alpha & numerics  No  No L - Full Yes Yes -
Fld #002 003 06 21 05 0006 U/C alpha & numerics  No  No L - Full Yes Yes -
Fld #003 004 07 21 05 0011 U/C alpha & numerics  No  No L - Full Yes Yes -  Fld #001
Fld #004 005 08 21 05 0016 U/C alpha & numerics  No  No L - Full Yes Yes - *Fld #003
```

Figure 6-5. Screen Field Documentation Showing Default Fields After Field Insertion.

Notice what has happened to the default field names in Figure 6-5. Fld #003, which previously defaulted to Fld #002, now defaults instead to Fld #001. That is because Fld #001, not Fld #002, is now the second field in the record. (Originally, Fld #002 was the second field.) Similarly, Fld #004, which previously defaulted to itself, now defaults to Fld #003 for the same reason - Fld #003, not Fld #004, is now the fourth field in the record.

To remedy this situation, it is necessary to invoke the Field Parameters Specification screen for Fld #003 and Fld #004 and change the default field names manually. This change should be made immediately after all necessary insertions or deletions have been completed.

6.6 SPECIAL CONSIDERATIONS FOR COMBINING FIELDS FROM SEVERAL FILES IN THE DISPLAY

IDEAS allows only one file to be associated with each screen. In many cases, however, it may be useful to combine fields from two or more files on one screen. In our master example, for instance, information from two files (a customer master file and a warehouse master file) is combined with some operator input to produce records for an invoice file. This situation requires special programming in the data entry module (see Chapter 8 for a discussion of the changes required) and special care in defining fields for the screen mask.

It is necessary to understand first that when several files are to be used by one screen, a complete record from each input file must be loaded into the work buffer. IDEAS will automatically load the record from the specified companion data file for this screen; records from any other files to be used must be loaded by a user-written subroutine (Chapter 8 discusses the procedure for coding such a routine.) The point is that the total length of all records from all files to be used as input cannot exceed the length of the work buffer (1008 bytes).

The second point to be emphasized is that the programmer must know the positions in the work buffer of all fields used on the screen which belong to files other than the companion file. Since the companion file record is the first to be loaded into the buffer and all other records will be concatenated to it, the position of a field in the second record loaded is computed by adding its position within its own record to the total record length of the first record in the buffer. The value thus obtained is the position which must be specified when the field parameters for that field are defined.

For example, suppose that a screen is to be defined which references two files - CUSTMSTR and WHSMSTR. CUSTMSTR is specified as the companion file for this screen; CUSTMSTR has records containing four fields with a total record length of 100 bytes. WHSMSTR has records containing 10 fields with a total record length of 200 bytes. All of the fields from CUSTMSTR records will be posted to the output file, but only two fields from WHSMSTR will be used, PROD. NO. and PROD. DESC. (See Figure 6-6).

Since CUSTMSTR is the specified companion file, its record is the first to be loaded into the buffer. The WHSMSTR record must be loaded into the buffer immediately following the CUSTMSTR record, by a user-written routine. Once both records have been loaded, the buffer looks like this:

```
Byte
1...............................100   101...................................148
CUST.                        DISCOUNT  PROD.                   RE-ORDER  PROD.
 NO.  NAME    ADDRESS           %       NO.   LOCATION   QTY     LEVEL    DESC.

      CUSTMSTR Record                                WHSMSTR Record
```

Figure 6-6. Work Buffer with Two Records.

Notice that PROD. NO. begins at position 101 in the buffer, while PROD. DESC. begins at 148. When the field parameters are defined for these fields, 101 and 148 must be entered as the field positions for PROD. NO. and PROD. DESC, respectively.

It can be seen from this example that the user must design all record formats for all files before attempting to create a screen mask which utilizes such records, since the positions of any fields other than those of the companion file must be known at the time the screen mask is created. Note, too, that if more than two files are used by a screen, the order in which records are loaded into the buffer will determine the positions of individual fields in the buffer.

CHAPTER 7
DATA ENTRY/INQUIRY/UPDATE PROGRAM GENERATION UTILITY

7.1 OVERVIEW OF THE DATA ENTRY/INQUIRY/UPDATE PROGRAM GENERATION UTILITY

Data entry, inquiry, and update programs are generated through the Data Entry/Inquiry/Update Program Generation Utility. These data entry programs allow data manipulation on files created through the Data Entry Utilities. The Data Entry/Inquiry/Update Program Generation Utility generates highly modularized code to facilitate user modification.

Eight different types of data entry programs can be generated, each with a different set of data entry operations. The user selects the desired type of data entry program. The available data entry programs are:

1. Inquiry only
2. Add a new record to the file
3. Add a new record or modify an existing one
4. Add a new record or delete an existing one
5. Add a new record or modify or delete an existing one
6. Modify an existing record
7. Delete an existing record
8. Modify or delete and existing record.

An inquiry only program displays information but does not allow modification.

A data entry, inquiry, or update program must be created for each data entry function desired, typically for each menu entry other than reports and the CANCEL key. The available data entry program options are divided into eight categories to allow the user to separate the data entry functions. Thus, certain data entry operations, such as modification or deletion of records, can be isolated from normal data entry routines. These "special" data entry operations may then be placed in a password-protected menu, eliminating tampering from unauthorized personnel. For example, it is reasonable to assume that certain payroll or personnel information should not be readily accessible for modification, deletion, or inquiry by unauthorized personnel. Sensitive information contained within a data record can be protected from unauthorized access by the use of different screen formats and data entry programs. These "sensitive" data entry programs should then be placed within password protected menus.

Data entry programs which access one data file typically will not need additional user modification. Conversely, data entry programs which access more than one data file will need additional user modification. The modification consists mainly of the addition of one or more system resident macros at a position indicated in the generated code. Some data entry programs will need modification to the generated code. Modification of data entry programs will be explained and exemplified later in this chapter. If modification will be required on data entry programs, it is important to specify the number of additional disk sectors needed for additonal application code when creating data entry programs. The Data Entry/Inquiry/Update Program Generation Utility then saves the program with the additional sectors specified.

7.1.1 The Data Entry/Inquiry/Update Program Generation Utility

Using the Data Entry/Inquiry/Update Program Generation Utility to create programs is very easy; the programs created are based on the type of data entry program required, and the specified screen and data file definitions. The user provides the file name for the program to be generated, the file name for the screen mask to be used, the file name for the data file to be accessed, selects the desired type of data entry program, specifies the number of fields required on the screen to establish the key, and specifies the number of disk sectors to be reserved if modification is anticipated.

The Data Entry/Inquiry/Update Program Generation Utility screen is pictured below with values entered from the master example. This screen will be referred to throughout this discussion of the Data Entry/Inquiry/Update Program Generation Utility.

```
IDEAS System - Data Entry/Inquiry/Update Program Generation Utility  Release 1.0
================================================================================
Create/revise date: JAN 24 80
                              File name for program to be generated:  "INQPROG2"
                              File name for screen mask to be used:   "INQSCRN1"
                              File name for data file to be accessed: "LOCATOR "

Functions to be provided in this program ( choose one from the list below ):  1

     1. ==] Inquiry only
     2.     Add a new record to the file
     3.     Add a new record or modify an existing one
     4.     Add a new record or delete an existing one
     5.     Add a new record or modify or delete an existing one
     6.     Modify an existing record
     7.     Delete an existing record
     8.     Modify or delete an existing record

Number of fields required on screen to establish the key to the data file:    1

Number of extra disk sectors to be provided for additional application code: 5#
```

Figure 7-1. The Data Entry/Inquiry/Update Program Generation Screen.

The file name for the data entry program used in this example is INQPROG2. The screen and data file to be used are INQSCRN1 and LOCATOR respectively. LOCATOR is an alternate key file associated with the Warehouse Master File. This is a type 1 program (inquiry only), and will display the specified information, but will not allow keyboard entry other than the key. The key, product number, will be entered by the operator. The information specified on INQSCRN1 will be displayed. This information may not be modified through this program. The number of fields needed on the screen to establish the key is 1.

This data entry program will perform an inquiry on the Warehouse Master File by product number. The Warehouse Master File contains information on the product number, location, and quantity only. To enhance the effectiveness of this inquiry, the data entry program will be modified to display certain important product information from the Product Master File. One additional sector is reserved on disk to allow modification of the program to display the product description and unit cost from the Product Master File. This modification will be discussed later in this chapter.

After all the fields in the screen have been entered as desired, RETURN to create the data entry program. To revise the data entry program, touch the edit key and use the RETURN key to skip to the field to be modified.

To revise a data entry program which was created previously, select the Data Entry/Inquiry/Update Program Generation Utility from the IDEAS menu, and enter the name of the data entry program to be revised. The system defaults to all the values previously entered. Touch the edit key and use the RETURN key to skip to the field(s) to be modified.

NOTE:

When the Data Entry/Inquiry/Update Program Generation Utility is used to revise a data entry program which was previously created, the data entry program is completely regenerated and all modifications made to the program are lost.

7.2 OVERVIEW OF DATA ENTRY/INQUIRY/UPDATE PROGRAM MODIFICATIONS

Modifications to created data entry programs are generally quite simple, consisting mainly of the addition of one or more system resident macros, or some minor modifications to the generated code. (A logical test, such as the value must be a multiple of five, is an example of a minor modification which must be made within the generated code.) The most common modifications to data entry programs are generally to allow access to more than one data file, or to use more than one screen per data entry program. Data entry programs which access one data file typically will not need additional user modification.

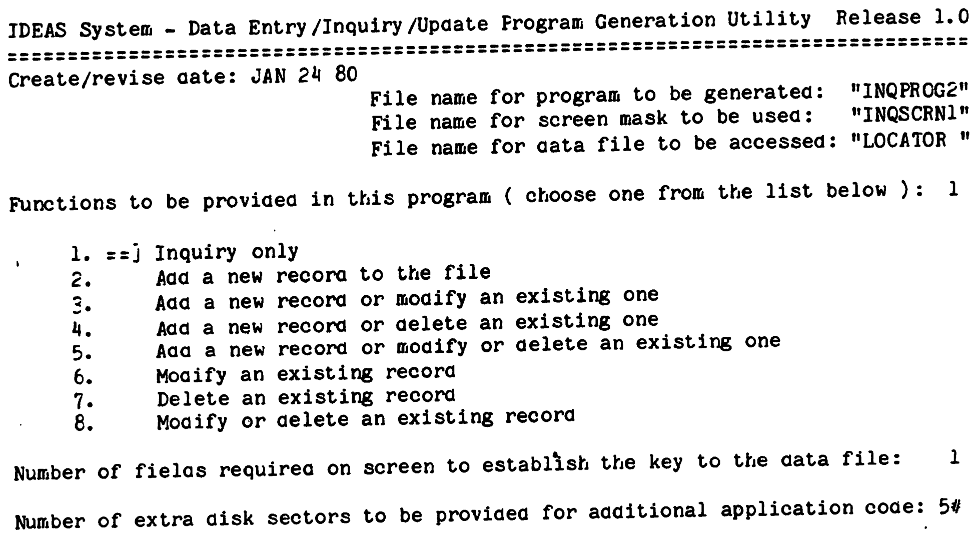
There are basically three different categories of data entry programs to be concerned with in modification: type 1 programs (inquiry only), type 2 programs (add a record only), and type 3-8 programs (add, delete, or modify -- in various combinations). To modify a type 1 program to accommodate use of additional data files or screens, the modifications are made in line 3980 (specified CODE). To modify a type 2 program to accommodate use of additional data files or screens, the modifications are made in the correct REM statements. For types 3-8, the modifications are made in the REM statements and in the specified CODE area(s).

Three modifications of data entry programs taken from the master example are explained in this section. The first example is the modification of a type 1 program to allow access to a second data file. The second example is the modification of a type 2 program to allow access to two additional data files. The third example is the modification of a type 5 program to allow access to a second data file.

7.2.1 Example 1: Modification of a Type 1 Program

This example uses INQPROG2 discussed earlier in this chapter. INQPROG2 is a type 1 program (displays the specified information, but does not allow keyboard entry other than the key). The key, product number, is entered by the operator. INQPROG2 performs an inquiry on the Warehouse Master File (WHRSMSTR) by product number. The Warehouse Master File contains information on the product number, location, and quantity only. To enhance the effectiveness of this inquiry, the data entry program is modified to display certain important product information from the Product Master File (PRODMSTR).

Since INQPROG2 is a type 1 program, the modifications occur in the area after line 3980 (specified CODE in the unmodified program). The screen used in creating INQPROG2 is provided below. A listing of the modified INQPROG2 follows the screen. The modifications of INQPROG2 are highlighted by brackets. This listing will be referred to throughout the discussion of the modification of INQPROG2.



```
IDEAS System - Data Entry/Inquiry/Update Program Generation Utility  Release 1.0
================================================================================
Create/revise date: JAN 24 80
                                  File name for program to be generated:  "INQPROG2"
                                  File name for screen mask to be used:   "INQSCRN1"
                                  File name for data file to be accessed: "LOCATOR "

Functions to be provided in this program ( choose one from the list below ):  1

     1. ==] Inquiry only
     2.     Add a new record to the file
     3.     Add a new record or modify an existing one
     4.     Add a new record or delete an existing one
     5.     Add a new record or modify or delete an existing one
     6.     Modify an existing record
     7.     Delete an existing record
     8.     Modify or delete an existing record

Number of fields required on screen to establish the key to the data file:    1

Number of extra disk sectors to be provided for additional application code: 5#
```

Figure 7-2. INQPROG2 Data Entry Program Creation Screen.

PROGRAM LISTING

```
1000 REM "INQPROG2" JAN 14 80 TYPE1
1010 IF O0$(9)<>"M" THEN 1020
     :LOAD DC T#2,"IDGLBSEL"1010,1010

1020 COM V0,F0$8
     :F0$="LOCATOR "
1030 GOSUB '32("INQSCRN1")
1040 REM Defaults
1400 GOTO 2000

1600 DEFFN'98
     :RETURN : REM FNKey Trap

1800 DEFFN'99
     :RETURN : REM Error Trap

2000 F=0
2010 F=F+1
2020 IF F>F0 THEN 4000
2030 GOSUB '34(F)
2050 ON FGOTO 2110,2120,2130,2140,2150,2160,2170
2110 REM "PRODNO  " - 001
     :GOTO 3900

2120 REM "PRODDESC" - 002
     :GOTO 2010

2130 REM "PRICE   " - 003
     :GOTO 2010

2140 REM "WHSLOC  " - 004
     :GOTO 2010

2150 REM "BAYLOC  " - 005
     :GOTO 2010

2160 REM "BAYPOS  " - 006
     :GOTO 2010

2170 REM "QUANTITY" - 007
     :GOTO 2010

3900 GOSUB '82(F0$)
3930 GOSUB '86(-V)
3960 GOSUB '141(F0$,K$(1),-1)
3970 IF Q<1 THEN 2030
3980 REM %
```

PROGRAM LISTING

CODE

```
    :GOSUB '141("PRODMSTR",K$,-101)
3990 GOSUB '36
4000 GOSUB '53("Touch EXEC for next inquiry, or CANCEL")
4040 GOSUB '34(129)
4070 IF Q<>32 THEN 4000
4170 GOTO 1030
```

The modification of INQPROG2 consists of the insertion of DEFFN'141 at line 3980. DEFFN'141 GETs and unpacks a record from a specified file. The general format of DEFFN'141 is:

```
DEFFN'141 (N$,K$,TO)
```

The following variables are used in this subroutine:

N$	The file name of the primary or alternate key file to be accessed by the data entry program.
K$	The key associated with the desired record.
TO	The record protect flag (- for non-protected records, + for protected records), and the position within the work buffer.

The position within the work buffer is the starting position for the field (to be accessed from the other data file) as listed in the screen documentation used in the data entry program. In this example, the position within the work buffer is 101, as is listed in the screen documentation for INQSCRN1. If a field which does not exist in any data file (such as a TOTALS field) is to be added to the screen, the position within the work buffer must be calculated by the user. The position within the work buffer must start after the final position of the last field in the associated screen. For example, if a screen contains fields up to byte 420, the position within the work buffer for a field which does not exist in any data file must begin at byte 421.

Refer to <u>Appendix A System Resident Macros</u> for more information on DEFFN'141.

7.2.2 <u>Example 2: Modification of a Type 2 Program</u>

This example uses the data entry program INVOPROG. INVOPROG is a type 2 program (allows new records to be added to the files, but does not allow modification or deletion of other records contained in the file). INVOPROG allows records to be added to the Invoice Master File (INVOMSTR) using INVOSCRN. The invoice number is the key associated with the Invoice Master File. The Invoice Master File contains information on the invoice number, customer number, ship to address, product number, description, and cost.

The Invoice Master File contains some fields found in the Product Master File (PRODMSTR) and the Customer Master File (CUSTMSTR). When INVOPROG is run, information is called in from the Product Master File and the Customer Master File, and this information is used to build the Invoice Master File. The information is called into the hidden fields in INVOSCRN, and then defaulted to fields within the Invoice Master File. INVOPROG, as created by IDEAS, must be modified to allow access of records from the Product Master File and the Customer Master File.

Since INVOPROG is a type 2 program, the modifications occur after the correct REM statements. The screen used in creating INVOPROG appears on the next page. A listing of the modified INVOPROG follows the screen. The modifications to INVOPROG are highlighted by brackets. This listing will be referred to throughout the discussion of the modification of INVOPROG.

```
IDEAS System - Data Entry/Inquiry/Update Program Generation Utility  Release 1.0
================================================================================
Create/revise date: JAN 24 80
                               File name for program to be generated:  "INVOPROG"
                               File name for screen mask to be used:   "INVOSCRN"
                               File name for data file to be accessed: "INVOMSTR"

Functions to be provided in this program ( choose one from the list below ):  2

    1.      Inquiry only
    2. ==] Add a new record to the file
    3.      Add a new record or modify an existing one
    4.      Add a new record or delete an existing one
    5.      Add a new record or modify or delete an existing one
    6.      Modify an existing record
    7.      Delete an existing record
    8.      Modify or delete an existing record

Number of fields required on screen to establish the key to the data file:    1

Number of extra disk sectors to be provided for additional application code: 5#
```

Figure 7-3. INVOPROG Data Entry Program
Creation Screen.

PROGRAM LISTING

```
1000 REM "INVOPROG" JAN 14 80 TYPE2
1010 IF O0$(9)<>"M" THEN 1020
     :LOAD DC T#2,"IDGLBSEL"1010,1010

1020 COM V0,F0$8
     :F0$="INVOMSTR"
1030 GOSUB '32("INVOSCRN")
1040 PRINT AT(4,3);BOX(4,26);AT(4,42);BOX(4,26)
1400 GOTO 2000

1600 DEFFN'98
     :RETURN : REM FNKey Trap

1800 DEFFN'99
     :RETURN : REM Error Trap

2000 F=0
2010 F=F+1
2020 IF F>F0 THEN 4000
2030 GOSUB '34(F)
2050 ON FGOTO 2110,2120,2130,2140,2150,2160,2170,2180,2190,2200,2210,2220,
     2230 ,2240 ,2250 ,2260 ,2270 ,2280 ,2290 ,2300 ,2310 ,2320 ,2330 ,2340
     ,2350 ,2360 ,2370 ,2380 ,2390 ,2400 ,2410 ,2420 ,2430 ,2440 ,2450 ,2460
      ,2470 ,2480 ,2490 ,2500 ,2510 ,2520 ,2530 ,2540 ,2550 ,2560 ,2570 ,
     2580 ,2590 ,2600 ,2610 ,2620 ,2630 ,2640
2110 REM "INVOICE#" - 001
     :GOSUB '56(D$,1)
     :GOSUB '46(2,D$(4))
     :F=2
     :GOTO 2010

2120 REM "DATE    " - 002
     :GOTO 2010

2130 REM "CUSTNO  " - 003
     :GOSUB '141("CUSTMSTR",K$,-344)
     :IF Q=0 THEN 2030
     :GOTO 2010

2140 REM "CUSTNAM " - 004
     :GOTO 2010

2150 REM "ADDRESS1" - 005
     :GOTO 2010

2160 REM "ADDRESS2" - 006
     :GOTO 2010

2170 REM "CITY    " - 007
     :GOTO 2010

2180 REM "STATE   " - 008
```

PROGRAM LISTING

```
    :GOTO 2010

2190 REM "ZIP      " - 009
    :GOTO 2010

2200 REM "DISC%    " - 010
    :GOTO 2010

2210 REM "SHPNAME " - 011
    :GOTO 2010

2220 REM "SHPADDR1" - 012
    :GOTO 2010

2230 REM "SHPADDR2" - 013
    :GOTO 2010

2240 REM "SHPCITY " - 014
    :GOTO 2010

2250 REM "SHPSTATE" - 015
    :GOTO 2010

2260 REM "SHPZIP   " - 016
    :GOTO 2010

2270 REM "QUAN#1   " - 017
    :X=Q
    :GOTO 2010

2280 REM "PROD#1   " - 018
    :GOSUB '141("PRODMSTR",K$,-474)
    :IF Q=0 THEN 2030
    :GOTO 2010

2290 REM "PRODESC1" - 019
    :GOTO 2010

2300 REM "DESC#1   " - 020
    :GOTO 2010

2310 REM "UNIT1    " - 021
    :GOTO 2010

2320 REM "PRICE#1 " - 022
    :X1=X*Q
    :GOSUB '55(23,X1)
    :GOTO 2010

2330 REM "COST1    " - 023
    :GOTO 2010
```

```
2340 REM "QUAN#2   " - 024
     :IF Q=0 THEN 2610
     :X=Q
     :GOTO 2010

2350 REM "PROD#2   " - 025
     :GOSUB '141("PRODMSTR",K$,-531)
     :IF Q=0 THEN 2030
     :GOTO 2010

2360 REM "PRODESC2" - 026
     :GOTO 2010

2370 REM "DESC#2   " - 027
     :GOTO 2010

2380 REM "UNIT2    " - 028
     :GOTO 2010

2390 REM "PRICE#2 " - 029
     :X1=X1+X*Q
     :GOSUB '55(30,X*Q)
     :GOTO 2010

2400 REM "COST2    " - 030
     :GOTO 2010

2410 REM "QUAN#3   " - 031
     :IF Q=0 THEN 2610
     :X=Q
     :GOTO 2010

2420 REM "PROD#3   " - 032
     :GOSUB '141("PRODMSTR",K$,-588)
     :IF Q=0 THEN 2030
     :GOTO 2010

2430 REM "PRODESC3" - 033
     :GOTO 2010

2440 REM "DESC#3   " - 034
     :GOTO 2010

2450 REM "UNIT3    " - 035
     :GOTO 2010

2460 REM "PRICE#3 " - 036
     :X1=X1+X*Q
     :GOSUB '55(37,X*Q)
     :GOTO 2010

2470 REM "COST3    " - 037
```

PROGRAM LISTING

```
     :GOTO 2010

2480 REM "QUAN#4   " - 038
     :IF Q=0 THEN 2610
     :X=Q
     :GOTO 2010

2490 REM "PROD#4   " - 039
     :GOSUB '141("PRODMSTR",K$,-645)
     :IF Q=0 THEN 2030
     :GOTO 2010

2500 REM "PRODESC4" - 040
     :GOTO 2010

2510 REM "DESC#4   " - 041
     :GOTO 2010

2520 REM "UNIT4    " - 042
     :GOTO 2010

2530 REM "PRICE#4 " - 043
     :X1=X1+X*Q
     :GOSUB '55(44,X*Q)
     :GOTO 2010

2540 REM "COST4    " - 044
     :GOTO 2010

2550 REM "QUAN#5   " - 045
     :IF Q=0 THEN 2610
     :X=Q
     :GOTO 2010

2560 REM "PROD#5   " - 046
     :GOSUB '141("PRODMSTR",K$,-702)
     :IF Q=0 THEN 2030
     :GOTO 2010

2570 REM "PRODESC5" - 047
     :GOTO 2010

2580 REM "DESC#5   " - 048
     :GOTO 2010

2590 REM "UNIT5    " - 049
     :GOTO 2010

2600 REM "PRICE#5 " - 050
     :X1=X1+X*Q
     :GOSUB '55(51,X*Q)
     :GOTO 2010
```

```
2610 REM "COSTS    " - 051
     :GOSUB '55(52,X1)
     :F=51
     :GOTO 2010

2620 REM "SUBTOT   " - 052
     :GOSUB '45(10)
     :GOSUB '55(53,X1*Q*.01)
     :GOTO 2010

2630 REM "DISCOUNT" - 053
     :GOSUB '45(10)
     :GOSUB '55(54,X1-X1*Q*.01)
     :GOTO 2010

2640 REM "TOTAL    " - 054
     :GOTO 2010

4000 GOSUB '53("Touch EXEC to accept, EDIT to modify")
4040 GOSUB '34(129)
4050 IF Q=240 THEN 2000
4070 IF Q<>32 THEN 4000
4080 GOSUB '142(F0$,1)
4090 IF Q<>0 THEN 1030
4100 INIT(09)E$
4110 STR(E$,35)="EDIT to modify, or CANCEL"
4120 GOSUB '35(E$)
4130 GOTO 4040
```

The modification of INVOPROG consists of six insertions of DEFFN'141 (occuring at lines 2130, 2280, 2350, 2420, 2490, and 2560) and five insertions of DEFFN'55 (occuring at lines 2320, 2390, 2460, 2530, and 2600. DEFFN'141 allows access of records in another data file, and DEFFN'55 computes the cost of a line item in the invoice and places this value in a specified field.

DEFFN'141 GETs and unpacks a record from a specified file. The general format of DEFFN'141 is:

DEFFN'141 (N$,K$,TO)

The following variables are used in this subroutine:

N$	The file name of the primary or alternate key file to be accessed by the data entry program.
K$	The key associated with the desired record.
TO	The record protect flag (- for non-protected records, + for protected records), and the position within the work buffer.

The position within the work buffer is the starting position for the field (to be accessed from the other data file) as listed in the screen documentation used in the data entry program. See the explanation in Example 1 for a more detailed discussion of the position within the work buffer.

Refer to Appendix A System Resident Macros for more information on DEFFN'141.

DEFFN'55 puts a specified value into the specified field. The general format of DEFFN'55 is:

DEFFN'55 (A, EO)

where A is the field number and EO is the value to be placed in field A.

Refer to Appendix A System Resident Macros for more information on DEFFN'55.

In the first modification, DEFFN'141 is added at line 2130. The fields in lines 2140 through 2200 are the fields to be read from the Customer Master File. The fields in lines 2210 through 2270 are the fields in INVOSCRN to which the fields from the Customer Master File are defaulted.

In the second modification, DEFFN'141 is added at line 2280. The fields in lines 2290 and 2320 are those to be read from the Product Master File. The fields in lines 2300 and 2320 are the fields in INVOSCRN to which the fields from the Customer Master File are defaulted. DEFFN'55 is added in line 2320 (in the second modification) to compute the cost of product # one. This value is then placed in field 23, COST1, at line 2330. Modifications three through six follow the same form and logic as the second modification. Since INVOSCRN contains five lines for different products on the invoice, it is necessary to repeat this modification for products one through five.

7.2.3 <u>Example 3: Modificiation of a Type 5 Program</u>

This example uses the data entry program WHRSPROG. WHRSPROG is a type 5 program (allows new records to be added to the files, and allows modification and deletion of other records contained in the file). WHRSPROG allows data manipulation on the records contained in the Warehouse Master File. The key to the Warehouse Master File is Product Number. The Warehouse Master File contains information on the product number, location, and quantity. To enhance the effectiveness of this data entry program, the program is modified to display certain important product information from the Product Master File (PRODMSTR).

Since WHRSPROG is a type 5 program, the modifications occur both after the correct REM statement and in the area after line 3980 (specified CODE in the unmodified program). The screen used in creating WHRSPROG is provided below. A listing of the modified WHRSPROG follows the screen. The modifications to WHRSPROG are highlighted by brackets. This listing will be referred to throughout the discussion of the modification of WHRSPROG.

```
IDEAS System - Data Entry/Inquiry/Update Program Generation Utility  Release 1.0
================================================================================
Create/revise date: JAN 24 80
                                File name for program to be generated:  "WHRSPROG"
                                File name for screen mask to be used:   "WHRSSCRN"
                                File name for data file to be accessed: "WHRSMSTR"

Functions to be provided in this program ( choose one from the list below ):  5

     1.      Inquiry only
     2.      Add a new record to the file
     3.      Add a new record or modify an existing one
     4.      Add a new record or delete an existing one
     5. ==]  Add a new record or modify or delete an existing one
     6.      Modify an existing record
     7.      Delete an existing record
     8.      Modify or delete an existing record

Number of fields required on screen to establish the key to the data file:   3

Number of extra disk sectors to be provided for additional application code: 5Ø
```

Figure 7-4. WHRSPROG Data Entry Program Creation Screen.

PROGRAM LISTING

```
1000 REM "WHRSPROG" JAN 14 80 TYPES
1010 IF OO$(9)<>"M" THEN 1020
    :LOAD DC T#2,"IDGLBSEL"1010,1010

1020 COM VO,FO$8
    :FO$="WHRSMSTR"
1030 GOSUB '32("WHRSSCRN")
1040 REM Defaults
1400 GOTO 2000

1600 DEFFN'98
    :RETURN : REM FNKey Trap

1800 DEFFN'99
    :RETURN : REM Error Trap

2000 F=0
2010 F=F+1
2020 IF F>FO THEN 4000
2030 GOSUB '34(F)
2050 ON FGOTO 2110,2120,2130,2140,2150,2160
2110 REM "WHSLOC   " - 001
    :GOTO 2010

2120 REM "BAYLOC   " - 002
    :GOTO 2010

2130 REM "BAYPOS   " - 003
    :GOTO 3900

2140 REM "PRODNO   " - 004
    :GOSUB '141("PRODMSTR",K$,-101)
    :IF Q=0 THEN 2030
    :GOTO 2010

2150 REM "PRODDESC" - 005
    :GOTO 2010

2160 REM "QUANTITY" - 006
    :GOTO 2010

3900 GOSUB '82(FO$)
3910 VO=V
3920 IF F$(V)<>" " THEN 2010
3930 GOSUB '86(-V)
3940 GOSUB '58(ABS(V),0,K$(1),0)
3950 IF Q=0 THEN 2010
3960 GOSUB '141(FO$,K$(1),1)
3970 IF Q<1 THEN 2030
3980 GOSUB '45(4)
    :GOSUB '141("PRODMSTR",K$,-101)
3990 GOSUB '36
```

PROGRAM LISTING

```
4000 E$="Touch EXEC to accept, EDIT to modify"
4010 IF F$(VO)=" " THEN 4030
4020 STR(E$,37)=", FN'9 to delete"
4030 GOSUB '53(E$)
4040 GOSUB '34(129)
4050 IF Q=240 THEN 2000
4060 IF Q=9 THEN 4140
4070 IF Q<>32 THEN 4000
4080 GOSUB '142(FO$,1)
4090 IF Q<>0 THEN 1030
4100 INIT(09)E$
4110 STR(E$,35)="EDIT to modify, or CANCEL"
4120 GOSUB '35(E$)
4130 GOTO 4040

4140 IF F$(VO)=" " THEN 4000
4150 INIT(20)Z$()
4160 GOSUB '142(FO$,0)
4170 GOTO 1030
```

The modification of WHRSPROG consists of two insertions of DEFFN'141: one at line 2140, the other at line 3980. DEFFN'141 GETs and unpacks a record from a specified file. The general format of DEFFN'141 is:

```
DEFFN'141 (N$,K$,TO)
```

The following variables are used in this subroutine:

N$	The file name of the primary or alternate key file to be accessed by the data entry program.
K$	The key associated with the desired record.
TO	The record protect flag (- for non-protected records, + for protected records), and the position within the work buffer.

The position within the work buffer is the starting position for the field (to be accessed from the other data file) as listed in the screen documentation used in the data entry program. See the explanation in Example 1 for a more detailed discussion of the position within the work buffer.

CHAPTER 8
REPORT/FORM PRINTING UTILITIES

8.1 OVERVIEW OF THE REPORT/FORM PRINTING UTILITIES

The Report/Form Printing Utilities are used to define a formatted report including several levels of headings and footers, control breaks, mathematical operations, and record selection criteria. Information required to define the report is stored in a report definition file. In addition to this file, the Report Utilities also generate BASIC code used to print the report using up to four different data files.

The report definition file contains specifications for both report format and report content. Format specifications incude the report title, page headers, and page footers, as well as field sequence and spacing. Content specifications include new fields (typically summation fields) for created reporting purposes, test criteria used to determine which records will appear on the report, a "sequence" file which will determine the sort order of records appearing on the report, and specifications for setting control breaks and totalling numeric fields.

The Report/Form Printing Utilities are provided to give the user the option of generating reports through a utility. Thus, the programming time needed to generate reports is eliminated. The sophisticated user may, however, desire to code his or her own customized reports. User-developed reports can be tied into IDEAS-based systems with a minimum of difficulty.

8.1.1 Definition of Terms Used in the Report/Form Printing Utilities

A number of terms referring to specific aspects of report formatting are used throughout this chapter and the Report/Form Printing utilities. Definitions of these terms are provided below for the convenience of the user.

Page header:	Is the report title specified by the user. The page header appears at the top of each page of the report.
	If line one of the report mask contains any data or text, the system assumes this to be the page header, and if a non-zero page length is specified, this line will be printed at the top of each new page.
	If the page header line contains the text MMDDYY or MMM DD YY, the date will be printed in that format in its place.
	If the page header line contains the text Page PPP, the page number will be printed.

MMDDYY and Page PPP are automatically provided by default on line one as well as the specified report title (page header). These must be deleted from the report mask if not desired.

<u>Group header</u> (optional): That set of line(s) which comprise a heading for a particular group (can consits of text and/or fields). The group header is printed once, and is printed again <u>only after a group field break or the beginning of a new page is encountered</u>. Group field breaks are inserted during report mask definition.

<u>Record header</u> (optional): A range of lines on the report mask consisting of text and/or fields that will be printed after the group header, then only after a record break field has been encountered. Record break fields are inserted during report mask definition.

<u>Record item:</u> A range of lines on the report mask consisting of text and/or fields that will be printed for each valid record found in processing the sequence key file and passing the logical record mask tests.

<u>Record footer</u> (optional): A range of lines on the report mask consisting of text and/or fields that will be printed when a record break field is encountered. After printing, any math fields will be set to zero (record level sub-totals). The record footer follows the record header in report mask definition.

<u>Group footer</u> (optional): A range of lines on the report mask consisting of text and/or fields which will be printed only when a group break field is encountered. After printing, any math fields in this range will be set to zero (group level sub-totals).

<u>Report footer</u> (optional): A range of lines on the report mask consisting of text and/or fields which will be printed at the end of the report (report totals). The record footer is specified during report mask definition.

<u>Page footer</u>: If a non-zero page length is specified, and if any text and/or fields are specified on the last two lines of the report mask, these will be printed on the last two specified lines of the page. Ex. - Assume a 132 col report (with a report mask length of 40 lines) and a specified page length of 60 lines. Lines 39 and 40 of the report mask will be printed at lines 59 and 60 of each page. Note: Math fields appearing on these lines will be set to zero after printing each page (page totals).

Math operations: Up to 32 math operations may be specified to be performed for each record processed. Each operation may be of the form:

N = (N2 + N3) + (N4 + N5) Dec = D

Where:	N1	= Field name
	N2 to N5	= Field names or constants
	"+" Represents any math operator (+ - * /)	
	D	= Decimal places

Constants: Up to 10 constants may be specified for math functions - up to 8 numeric characters (0 to 9 and +, =, and decimal point).

Default key range: A defaut low and high key may be specified as limits on the sequence file. These may be optionally modifiable by the user at run time.

Logical record masks: Up to 4 fields may be tested for each record read. Only those records which meet all specified conditions will be included in the report. For each test, specify:

a. Field Name
b. Test (= , , |, =, |=, |)
c. Test Value (up to 64 characters).

Both the logical record masks and the default field values may be made optionally modifiable at run time, either separately or combined.

8.2 REPORT/FORMS PRINTING UTILITIES: CREATION/REVISION

Report creation and revision follow the same format and use the same screens, the only difference being that when a report is revised, the user enters the report name, and the report screens appear wih all the values for the report to be revised entered.

The report program module screen is pictured below with values entered from the master example. This screen will be referred to throughout the discussion of the Report/Forms Printing Utilities.

```
IDEAS System Utility - Report Program Module for "WREP0000"         Release 1.0
================================================================================
Report Title: "Herman Melville Co. Warehouse Report                           "
Sequence File: "WHRSMSTR"  Page width (col) =  80 Length (rows) = 66 Date 012480
Additional files:    File Name    Key Field    Constants (CONSTNT0,etc) C0=1
                     "PRODMSTR"/"PRODNO  "      C1=0          C2=0         C3=0
                     "          "/"      "      C4=0          C5=0         C6=0
                     "          "/"      "      C7=0          C8=0         C9=0
Default low key  =   "                                                        "
Default high key =   "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz"
Field Name Op Mask: Process only records which meet the following =,],]=,[,[=,[]
"         "   "                                                               "
"         "   "                                                               "
"         "   "                                                               "
"         "   "                                                               "
Run time options: ( 0=none, 1=default key limits only, 2=masks only, 3=both )  3
 Report Detail  Group    Record  Record  Record  Group   Report          Break
(0=none 66=max) Header   Header   Item   Footer  Footer  Footer          Fields
1st line            2        6     10      13      16      20    Record "BAYLOC  "
Last line           5        9     12      15      19      22    Group  "WHSLOC  "
Math Functions: Current function is #  3 of 32 ,    ( Operators: + , - , * , / )
"TOTAL3  " = ("TOTAL3  " + "CONSTNT0")   ("        "  "        ") Dec places: 0
NOTE: At any time - FN'1=Mask edit, FN'2 = Field names, FN'3 = Math, FN'20 = End
EDIT = modify, FN'1 = mask, FN'2 = fields, FN'3 = math, FN'20 = end            #


Herman Melville Co. Warehouse Report                   MMM DD YY          Page PPP

Warehouse Number ##
===================

     Bay Location ##
     ---------------
Bay Pos     Product#     Quantity    Description                        Unit Cost
================================================================================

   ##       ########       ####      ##############################     #########


     Total number of different products in this bay area  ###


Total number of different products in this warehouse  ###



Total number of different products ###


[Row 21 Col 035] FN'0 = Define/Edit field,  FN'20 = Return to main module
```

Figure 8-1. The Report Program Module Screens.

The example used in Figure 8-1 is taken from the master example. This report, WREP0000, is a warehouse report generated for the Herman Melville Co. which allows the company to keep track of the warehouses.

8.3 REPORT CREATION/REVISION

Select the operation desired (creation/revision/documentation)

For creation/revision, follow the instructions to get to the appropriate file. Note: the file name for the report program MUST contain at least one uppercase letter.

Enter the report title and the file name for the primary or alternate key file which is to supply the "sort" order for the records in the report. Then, enter the page width in columns (80-158). Note: not all printers will support all possible page widths.

The maximum number of definable report mask lines will be displayed on line 18 of the screen. This has no bearing on the possible number of lines to be specified as the page length. It mereltells you how many different lines may be used to define the various portions of the report.

At any time after the page width has been specified, the report mask may be created and/or edited by touching FN'01. This will bring you to the report mask editor. Touching FN'20 while in the report mask editor will return you the current maodule at the same field as when FN'01 was touched.

Enter any additional files to be accessed by the report. Note: a maximum of three additional files may be used. The system does NOT check to be sure that the 128 field total has been reached, nor does it check to be sure that the 1008 byte total buffer size has not been exceeded. That remains the user's responsibility. This also pertains to fields which are created in the report mask editor.

For each additional file, a default key field name is supplied. THIS NAME IS PROBABLY NOT CORRECT! It is name of the key field as specified in the file just named. What you want is the name of the field in the sequence file, or a previously defined additional file which will supply the key to access the current file. Often the field names will be different, although they may be the same.

If you do not remember the field names, and do not have the file documentation handy, you may review the currently defined field names by touching FN'02.

Enter any math constants desired for computations in the report module (up to 10 constants may be entered). Next, enter the default values for the lowest and highest keys to processed in the sequence file. (Default values are supplied by the program which will cause the entire file to be processed. Touch EXEC in both cases to accept the system-supplied default values.)

Enter up to four fields to be checked in each record processed. If a field name is entered, a cmparison operator and a test value are required. If any tests are specified, only those records which pass <u>all</u> of the specified tests will be processed.

Specify the run-time options:

0 - Means there are no run-time options
1 - Means the default low and high sequence file keys default values may be modified by the operator at run time.
2 - Means the four field record tests may be modified by the operator at run time.
3 - Means that either of the above may be modified by the operator at run time.

Specify the starting and ending line numbers on the report mask which define each of the following sections of the report to be used.

A. Group header
B. Record header
C. Record item

> NOTE:
>
> If "form filling" rather than "report" format is to be used, record item is the only parameter to be used, and must start at line 1 and end at line N where N = the number of lines available for definition in the report mask.

D. Record footer
E. Group footer
F. Report footer

> NOTE:
>
> A zero for the starting line number in each of the above will cause a zero to be entered automatically as the ending line number and that parameter will not be used in the report.
>
> The report mask should have been defined before the above have been specified. The mask may be created or editied at any time after the page width has been input by touching FN'01.

Specify the record and group break fields, if any. The break field is a field which causes the record and/or group footer to be printed when its value changes in a new record.

Define any desired math functions as follows:

A. Touch FN'03

B. Enter the number of the math function (1-32). The math functions must be in order starting with number 1.

C. Enter the destination field name (this must be a defined field. It may have been previously defined in one of the data files used by the report, or it may a field that has been newly defined in the report mask editor.)

D. Enter the remaining field names (or constant names) and the desired operators. A blank field name iwll complete the math operation if 4 operation fields are not input. To use one of constants defined above, the "field name" is CONSTNT0 - CONSTNT9.

E. Enter the desired number of decimal places (this will be a default value from the destination field parameter, and will cause the result to be rounded to the specified number of decimal places.)

8.4 REPORT MASK DEFINITION

To define and/or edit the report mask, touch FN'01 at any time after the page width has been entered. The first line on the report mask will automatically have the report title starting at the left margin. It will have the date specification (MMDDYY) and the page number specification (Page PPP) ending at column 80.

If you do not want any of these items, delete them from the line. If the "Page PPP" is left anywhere on the first line, and if the "form filling" format is not used, and if there is a specified page length, the page number will be printed on the first line of each new page in the report.

Similarly, the date will also be printed if the date specification is left on the first line, and all of the other conditions specified into the previous paragraph are true. The date may be specified in either of 2 ways (Ex. August 27, 1979):

1. MMDDYY formt (default format)	082779
2. MMM DD YY format	AUG 27 79

The entering of text on the report mask is very similar to the entering of text in the screen definition module (qv). ie., all of the EDITING keys operate, etc. The primary differences in the text entry between the report mask editor and the screen mask editor are that:

1. The screen scrolls to handle greater than 80 columns
2. The screen scrolls vertically to handle more than 24 lines
3. The "END" key, (FN'04) moves the cursor to row R, column C where R is the maximum number of lines that may be defined and C is the page width.

The definition of fields, however, varies significantly from thescreen generation routines.

To return to the report definition module from the report mask editor, touch FN'20. If you have inadvertantly entered the field definition module, edit a field then return to the mask editor and touch FN'20.

To define a field in a report:

1. Position the cursor at the row and column on the report mask where you want the field to begin and touch FN'0 (only when in the report mask editor)

2. A list of all currently defined fields will be displayed in row/column format. Those fields already used will be underscroed. (a field may only be used once in a report)

3. To use an already defined field, enter the row and column (where the field appears in the field list, NOT the row and column on the report mask definition screen). This is necessary instead of entering the field name because of the possibility if having several fields with the same name from different files.)

> NOTE:
>
> It is important to remember the order in which the files were specified if you are using more than one file. The source files are not specified int the list of field names, but the fields are listed in alphabetic order by file with the files in the order in which they were specified. Newly defined fields appear at end of the list and may not be in alphabetic order.

4. The field name, length, number of decimal places, and row & column on the report mask will appear. EXEC through these fields (modifying any that you desire). After this has been completed, the system will automatically return to the report mask editor.

5. To review/revise a field already specified for the report, position the cursor anywhere in the field and touch FN'0, then return to step 4 above.

6. To define a new field, execute step 1 above, then touch EXEC (or enter "0") when the cursor is in the ROW field. This will cause the next available position in the list to become the new field that is deing defined. Step through the remaining parameters as in step 4 above, but entering a field name, length, and number of decimal places. A numberic field should not exceed 13 bytes in length.

When you have completed the report mask editing and have returned to the report difinition module, and it has been completed, touch FN'20 to end the report definition. This will return you to the report menu, and you may document the report if you desire.

The Herman Melville Co. Warehouse Report is pictured below with certain salient points highlighted by additional text.

```
Herman Melville Co. Warehouse Report              JAN 24 80        Page 001

Warehouse Number 01
===================

     Bay Location AA
     ---------------
Bay Pos      Product#      Quantity    Description                           Unit Cost
==========================================================================================

  01         00000001         25       Still life with pumpkin                    9.95

  10         00000002         10       Folding butcher block table              249.95

     Total number of different products in this bay area     2

     Bay Location BB
     ---------------
Bay Pos      Product#      Quantity    Description                           Unit Cost
==========================================================================================

  01         00000007         50       RK201-T "REDHEAD" 3.5in high              23.95

  02         00000008         75       RK202-T "BACK TO SCHOOL" 5.5in            23.95

  03         00000009          3       RK203-T "CARROLER" 5.5in high             24.95

     Total number of different products in this bay area     3

Total number of different products in this warehouse     5


Warehouse Number 02
===================

     Bay Location AA
     ---------------
Bay Pos      Product#      Quantity    Description                           Unit Cost
==========================================================================================

  01         00000003         50       Solid c. hurricane candle hldr            15.95

  02         00000004       1000       Six copper napkin rings                   14.95

  03         00000005       2000       Fringed napkins                            9.95

     Total number of different products in this bay area     3

Total number of different products in this warehouse     3

Herman Melville Co. Warehouse Report              JAN 24 80        Page 002
Warehouse Number 03
===================

     Bay Location CC
     ---------------
Bay Pos      Product#      Quantity    Description                           Unit Cost
==========================================================================================

  01         00000006         50       Initialized wine glass                    12.95

     Total number of different products in this bay area     1

Total number of different products in this warehouse     1


Total number of different products     9
```

Figure 8-2. The Herman Melville Co. Warehouse Report.

CHAPTER 9
IDEAS SUPPLEMENTARY DATA FILE UTILITIES

9.1 OVERVIEW

The IDEAS Supplementary Data File Utilities are contained on the IDEAS application utilities diskette. The IDEAS Supplementary Data File Utilities are a group of utility programs provided with IDEAS designed to accent the functionality of IDEAS generated applications. The utilities consist of a menu module, which can be interfaced with any IDEAS generated applications and, then, accessed from a menu created through the Application Menu Program Utilities, and utility programs which provide specific user functions. These functions include check file status (percent full, ect.), protect or release all records in a file (2200MVP and 2200VP only), reconstruct the key files, and convert to or from Wang standard telecommunications file format. The menu module and utilities must be copied to specific device addresses selected by the user in the "IDEAS System Utility - Application Device Selection Module". These device numbers, and neccesary program and data files are identified in Section 9.2, IDEAS SUPPLEMENTARY DATA FILE UTILITIES INSTALLATION PROCEDURES.

9.1.1 IDEAS Supplementary Data File Utilities Menu Module

The Supplementary Data File Menu Module consists of one application program, IDFU-3M0, and two data files, idfu-3m0 and idfu-3m1. The menu module is hardware dependent and will display the proper menu for either 2200T or 2200MVP and 2200VP. Four utilities are available for the 2200T. These are:

Check File Status
Reconstruct key file(s)
Convert IDEAS data file to Wang Telecommunications file format
Convert Wang Telecommunications file to IDEAS data file format

Two additional utilities are available for the 2200VP and MVP. These utilities are:

Protect all records in a file
Release all records in a file

The Supplementary Data File Utilities menu display may be modified through the Application Menu Program Utilities to include other user developed utilities or password security. For a complete discussion of the Application Menu Program Utilities see Chapter 6, Application Menu Program Utilities. For these special menu modules it is not necessary for the user to associate a program name or load message with the "Cancel" key. The function key (FN') defined by the user as the "Cancel" key, in the Application Initialization Utility Module, will automatically exit to the menu from which the Supplementary Data File Utilities Menu Module was initially loaded.

> NOTE
>
> Before user modification of the 2200T menu module, idfu-3ml, a dummy menu application program IDFU-3M1 must be created. This can easily be accomplished by copying and renameing the menu module application program IDFU-3M0.

9.1.2 Check File Status

The Check File Status utility was designed to provide IDEAS applications users with an to easy to use inquiry application which provides important data file status information. This utility can be accessed by depressing FN' 00 of the Supplementary Data File Utility Menu Module. An input screen is displayed and the user is prompted to enter a filè name for the data file to be reviewed. If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file does not exist or has not been opened by the "START" module, an error message is displayed and the user has the option to enter a different file name, or exit from the utility (SF'31). If a valid data file name has been entered a second utility screen then displays the following information:

Number of records specified	The number of records specified for the data file in the IDEAS Data File Performance Option Selection Module of the Data File Utilities.
Number of records provided	The number of records specified + n% distributed free space. (nominally, n = 5%, however this may be different dependent on key length)
Number of records now used	The number of data records in the file.
Number of records available	The number of records which can be added to the file before 100% Full (actual) occurs.
% Full (as specified)	The percent full as based on the number of records specified by the user during file creation.
% Full (actual)	The percent full based on the actual number of records provided for during file creation.

Number of overflow records	The number of keys which have overflowed from one bucket to another.
Overflow percentage	A calculation based on, Number of overflow records / Number of records now used * .01
Maximum possible number of records/bucket:	A calulation based on, Number of records provided / Number of buckets.
Maximum actual current records/bucket:	The largest amount of records contained in any one bucket of the file.
Minimum actual current records/bucket:	The smallest amount of records contained in any one bucket of the file.
Average actual current records/bucket:	A calculation based on, b1+b2+b3+bn... / Number of buckets in the file (Where b = The number or records in each bucket).

When all the file status information has been displayed the user has the option of returning to the Supplementary Data File Utilities Menu Module by pressing RETURN or exiting to the menu from which the Supplementary Data File Utilities Menu Module was loaded by touching the FN key specified as the "Cancel" key during the "START" program generation.

9.1.3 Protect All Records in a File (2200MVP & 2200VP)

The Protect all records in a file utility was designed to provide the user with a way to protect all records in a file at the record level by the "using" partition or CPU. This utility is extremely useful in that access of existing records in a file can be limited to one partition, in 2200MVP configurations , or one CPU in multiplexed 2200VP configurations. For information concerning record protection see Chapter 4, Data File Utilities.

The utility can be accessed by pressing SF' 01 of the Supplementary Data File Utilities Menu Module. An input screen is displayed and the user is prompted to enter a file name for the records to be protected. If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file does not exist or has not been opened by the "START" module, an error message is displayed and the user has the option to enter a different file name, or exit from the utility by pressing the SF' key specified as the "CANCEL" key in the "START" program generation module.

If a valid data file name has been entered the utility then displays the number of records provided for in the file. The user may continue and protect the records by keying RETURN, or cancel by touching the SF key specified as the "Cancel" key. Successful execution of this utility returns the user to the Supplementary Data File Utilities Menu Module, cancellation will return the user to the menu from which the Supplementary Data File Utilities Menu Module was loaded.

9.1.4 Release All Records in a File (2200MVP & 2200VP)

The Release all records in a file utility was designed as a counterpart to the Protect all records in a file utility. The utility, when executed, will "turn off" the record protect byte set by the Protect all records in a file utility. With the record protect byte "turned off" file access is allowed from any calling partition or CPU.

This utility can be accessed by pressing FN' 02 of the Supplementary Data File Utilities Menu Module. An input screen is displayed and the user is prompted to enter a file name for the records to be released. If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file does not exist or has not been opened by the "START" module, an error message is displayed and the user has the option to enter a different file name, or exit from the utility by pressing the SF' key specified as the "CANCEL" key in the "START" program generation module.

If a valid data file name has been entered the utility then displays the number of records provided for in the file. The user may continue and protect the records by keying RETURN, or cancel by touching the SF key specified as the "Cancel" key. Successful execution of this utility returns the user to the Supplementary Data File Utilities Menu Module, cancellation will return the user to the menu from which the Supplementary Data File Utilities Menu Module was loaded.

9.1.5 Reconstruct Key File(s)

The Reconstruct key file(s) utility was developed to provide end users with a procedure by which damaged key files can be rebuilt quickly and easily. The utility;

Initializes any alternate key files.
Initializes the primary key file index.
Determines primary key composition, length, field position within the data record, and sort order from information contained in the primary key file data file definition
Determines any alternate key composition, length, field position within the data record, and sort order from information contained in the alternate key file data file definition
Reads the existing data records and reconstructs the primary data file index for complete records or deletes partial records
Reconstructs associated alternate key files

The reconstruct key file(s) utility can be accessed by pressing FN' 03, for 2200MVP & 2200VP, or FN' 01, for 2200T, of the Supplementary Data File Utilities Menu Module. An input screen is displayed and the user is prompted to enter a file name for the key file to be reconstructed. The user must enter a PRIMARY FILE name, an alternate key file cannot be used for reconstruction. Alternate key files contain only keys and pointers to the data records in the primary data file.

If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file does not exist or has not been opened by the "START" module, an error message is displayed and the user has the option to enter a different file name, or exit from the utility by pressing the SF' key specified as the "CANCEL" key in the "START" program generation module.

A second utility screen is displayed containing the names of the primary key file and its associated alternate key files (if any exist) that are to be reconstructed. The user may continue key file reconstruction by touching RETURN, or abort the procedure by pressing the FN key defined as the "Cancel" key. Successful execution of this utility will return the user to the Supplementary Data File Utilities Menu Module, cancellation returns the user to the menu from which the Supplementary Data File Utilities Menu Module was loaded.

NOTE

If the actual data records are damaged, reconstruction will delete keys and data for the damaged records. Damaged data records can not be reconstructed. If relative Sector 0 of the primary data file, which contains data file parameters, is damaged the key file reconstruction utility will apend.

It is strongly suggested that BACKUP copies of data file definitions and primary data files be kept. BACKUP copies of primary data files can easily be created and maintained by using the supplementary data file utility - Convert IDEAS data file to Wang Telecommunications file format. The data records in these BACKUP copies can then be easily restored to the online files by using the utility - Convert Wang Telecommunications file to IDEAS data file format.

9.1.6 Convert IDEAS Data File to Wang Telecommunications File Format

The Convert IDEAS data file to Wang Telecommunications file format utility was designed and implemented to provide the user with an easy to use conversion process by which IDEAS HIKAM files may be converted into Wang standard telecommunications format for subsequent data transmission. Files converted from IDEAS data file format to Wang telecommunications file format may be transmitted to host or remote sites using any of the Wang Telecommunications Emulators.

This conversion utility can be accessed by depressing SF' 04, for 2200MVP & 2200VP, or SF' 02, for 2200T, of the Supplementary Data File Utilities Menu Module. A data entry screen is displayed and the user is prompted to enter;

A valid input IDEAS HIKAM file name

If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete the entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file does not exist or has not been opened by the "START" module, an error message is displayed and the user has the option to enter a different file name, or exit from the utility by pressing the SF' key specified as the "CANCEL" key in the "START" program generation module.

An output Telecommunications file name

If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character.

An output address for the Telecommunications file

Entry is automatically terminated on the third character. If an invalid or unattached device is specified, an error message is displayed and the user must reenter a valid device selection or exit from the utility by pressing the FN" key defined as the "CANCEL" key. If the ouput file name entered currently exists on the output device selected a warning message is displayed and the user may touch RETURN to abort utility execution or enter NO to the prompt and press RETURN to continue. If the user aborts the utility at this point the parameters are cleared and the user is returned to the first input fiels of the data entry screen. If the user continues the the number of sectors to be reserved for the output file will default to the number selected when the file was originally created. The TC file created will overwrite the previously catalogued TC file.

The number of sectors to be reserved for the output file

If the entry is less than five digits, the user must enter a RETURN when the entry is complete in order to complete this entry; if the entry is a full five digits, entry is automatically terminated on the fifth character. If user chooses to overwrite a previously created TC file, this parameter will default to the number of sectors previously saved on the output platter.

Whether or not output records are to be concatenated

Non-concatenated records are padded with spaces at the end of the output record to produce 80 character records. If the input record is longer than 80 characters the output records will be multiples of 80. On the other hand concatenated records are not padded, input records are written to 80 character records with one record starting at the end of another record. For example suppose we have a 65 character input record. IF these records are concatenated, the first record of the input file will be written to the first 65 bytes of the output record, the second input record would then be written to the last 15 bytes of the first output record and first 50 bytes of the second output record. If these records had not been concatenated each 65 byte input record would be padded to 80 bytes. Input record one would be ouput record one, input record two would be output record two and so on.

If an invalid selection is specified an error message is displayed and the user must reenter a valid selection or exit from the utility by pressing the FN' key defined as the "CANCEL" key.

Up to four logical tests for input records to be processed

The choice of using logical field tests is optional. Stringing of logical tests is restricted to the "AND" conjunction only. If the field name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the field does not exist, in the record, an error message is displayed and the user has the option to enter a different field name. The user then enters a mathamatical operator, if the operator is less than two characters, the user must enter a RETURN when the entry is complete; if the entry is a full two characters, entry is automatically terminated on the second character. If the operator is invalid an error message is displayed and the user has the option to correct the entry.

After entering the field to be tested and the operator, the user then enters a value by which to test the field within each record. This value should match the format and length of the field parameters in the record. For example if the user wishes to test FIELD001, which is five characters long, numeric only and zero filled. The test value to write output records where FIELD001 is greater than zero must be 00000.

NOTE

All records should be released (unprotected) before converting to Telecommunications format. Protected records cannot be written when converting back to IDEAS file format. Alternate or primary key file names may be entered for the input file.

After all necessary information has been entered, the user can continue by pressing RETURN, or cancel by touching the FN key specified as the "Cancel" key. Successful execution of this utility will return the user to the Supplementary Data File Utilities Menu Module, cancellation will return the user to the menu from which the Supplementary Data File Utilities Menu Module was loaded.

9.1.6 Convert Wang Telecommunications File to IDEAS Data File Format

The Convert Wang Telecommunications file to IDEAS data file format utility was designed as a counterpart to the Convert IDEAS data file to Wang Telecommunications file format utility. Files previously converted from HIKAM to Telecommunications format can be easily converted back to IDEAS HIKAM file format to update online data files.

The utility can be accessed by depressing FN' 05, for 2200MVP & 2200VP, or FN' 03, for 2200T, of the Supplementary Data File Utilities Menu Module. A data entry screen is displayed and the user is prompted to enter;

A valid output IDEAS HIKAM file name

If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character. If the file does not exist or has not been opened by the "START" module, an error message is displayed and the user has the option to enter a different file name, or exit from the utility by pressing the FN' key specified as the "CANCEL" key in the "START" program generation module.

An input Telecommunications file name

If the file name is less than eight characters, the user must enter a RETURN when the name is complete in order to complete this entry; if the name is a full eight characters, entry is automatically terminated on the eighth character.

An input address for the Telecommunications file

> Entry is automatically terminated on the third character. If an invalid or unattached device is specified an error message is displayed and the user must reenter a valid device selection or exit from the utility by pressing the SF' key defined as the "CANCEL" key. If the input file name entered does not exist on the device selected an error messgae is displayed and the user may touch reenter a valid input file name or exit from the utility by pressing the FN' key specified as the "CANCEL" key in the "START" program generation module.

Whether or not input records are concatenated

> It is imperative that the correct selection be made concerning telecommunications record concatenation. If the wrong selection is made the output data file may be damaged, or incorrectly formatted records may be written into the output file.

After all necessary information has been entered, the user may continue by keying RETURN, or cancel by touching the FN key specified as the "Cancel" key. Successful execution of this utility will return the user to the Supplementary Data File Utilities Menu Module, cancellation will return the user to the menu from which the Supplementary Data File Utilities Menu Module was loaded.

9.2 IDEAS SUPPLEMENTARY DATA FILE UTILITIES INSTALLATION PROCEDURES

Certain restrictions of the BASIC language on the 2200T has made some of the utilities hardware dependent. Installation of the Supplementary Data File Utilities is relatively simple, however the following steps should be followed.

9.2.1 2200T Installation Procedures

1. Copy the following program files to the device address selected as Device # 02 in the I.D.E.A.S. System Utilities - Application Device Selection Module.

   IDFU-310 - 2200T Device Selection Module
   IDFU-311 - 2200T Device Selection Module

2. Copy the following data files to the device address selected as Device # 03 in the I.D.E.A.S. System Utilities - Application Device Selection Module.

ideas302 -	Companion screen file IDFU-300, IDFU-302
ideas315 -	Companion screen file IDFU-300
idfu-302 -	Companion screen file IDFU-302
idfu-303 -	Companion screen file IDFU-312
idfu-304 -	Companion screen file IDFU-314
idfu-3m1 -	Companion data description IDFU-3MO

3. Copy the following program files to the device address selected as Device # 04 in the I.D.E.A.S. System Utilities - Application Device Selection Module.

IDFU-300 -	Check File Status
IDFU-302 -	Reconstruct key file(s)
IDFU-312 -	Convert IDEAS data file format to Wang Telecommunications file
IDFU-313 -	Convert IDEAS data file to Wang Telecomunications file format
IDFU-314 -	Convert Wang Telecommunications file to IDEAS data file format
IDFU-315 -	Convert Wang Telecommunications file to IDEAS data file format
IDFU-3MO -	IDEAS Supplementary Data File Utilities Menu Module

9.2.2 2200MVP & 2200VP Installation Procedures

1. Copy the following data files to the device address selected as Device # 03 in the I.D.E.A.S. System Utilities - Application Device Selection Module.

ideas302 -	Companion screen file IDFU-300, IDFU-302
ideas315 -	Companion screen file IDFU-300
idfu-302 -	Companion screen file IDFU-302
idfu-303 -	Companion screen file IDFU-312
idfu-304 -	Companion screen file IDFU-314
idfu-3m0 -	Companion data description IDFU-3MO

2. Copy the following program files to the device address selected as Device # 04 in the I.D.E.A.S. System Utilities - Application Device Selection Module.

IDFU-300 -	Check File Status
IDFU-301 -	Protect all records in a file / Release all records in a file
IDFU-302 -	Reconstruct key file(s)
IDFU-303 -	Convert IDEAS data file format to Wang Telecommunications file
IDFU-304 -	Convert IDEAS data file to Wang Telecomunications file format
IDFU-305 -	Convert Wang Telecommunications file to IDEAS data file format
IDFU-306 -	Convert Wang Telecommunications file to IDEAS data file format
IDFU-3MO -	IDEAS Supplementary Data File Utilities Menu Module

APPENDIX A
SYSTEM RESIDENT MACROS

All IDEAS based systems make use of the system resident macros. The system resident macros are a set of very powerful subroutine calls. The system resident macros are contained in two programs: ID-SUB-T, for use with the 2200T and VP, and ID-SUB-M, for use with the 2200MVP.

Both programs contain the same set of subroutines, but ID-SUB-M is used as a global subroutine module, while ID-SUB-T is loaded with the actual application progams. The user does not have to worry about loading ID-SUB-T, as it is automatically loaded by the IDEAS-generated "START" modules, and overlays are performed. ID-SUB-M is automatically loaded by IDEASVAR when IDEAS-based applications are loaded.

The system resident macros may also be used to modify the code generated by IDEAS. Use of the system resident macros can greatly uncrease the power of IDEAS-based systems.

LIST OF SUBROUTINES AVAILABLE IN IDEAS SYSTEM SUBROUTINE MODULES

DEFFN'32(N$)	Get field parameters and display screen.
DEFFN'33(N$)	Get field parameters for screen N$, do not display screen.
DEFFN'34(F)	Allow input from keyboard into field number "F".
DEFFN'35(E$)	Display E$ as error message on line 24, sound audio alarm.
DEFFN'36	Display all fields starting at current field # "F".
DEFFN'37(Q)	Display field # "Q".
DEFFN'38(N$)	Get field parameters, display screen, keep current record.
DEFFN'39(N$,P)	Get limits of file "N$" on device # "P".
DEFFN'40(N$,E$)	Load program "N$", display loading message "E$".
DEFFN'41(V,K4,TO)	GET and unpack record from file number "V".
DEFFN'42(V,P)	Pack record and PUT it in file number "V".
DEFFN'44(R,C,K$,L)	Display "L" bytes of "K$" at row "R", column "C".
DEFFN'45(Q)	Retrieve contents of field number "Q".
DEFFN'46(Q,K$)	Put contents of "K$" into field number "Q".
DEFFN'47(P,K$,L)	Copy "L" bytes to/from "K$" from/to record position "P".
DEFFN'48(C,K$(1),L)	Copy "L" bytes of "K$(1)" to print buffer at position "C".
DEFFN'49(L)	Print "L" bytes of the print buffer, reset buffer.
DEFFN'50	Reset print buffer.
DEFFN'51(Q,M)	Copy field # "Q" to print buffer at screen column + "M".
DEFFN'52(Q)	Unpack all field parameters for field number "Q".
DEFFN'53(E$)	Display "E$" on line 24 as message to operator.
DEFFN'54(P,K$,(1))	Internal routine used in date validation routines.
DEFFN'55(A,EO)	Round value "EO" to field spec. dec., put in field # "A".
DEFFN'56(K$,Q)	Date validation/conversion subroutine.
DEFFN'57(L)	Convert Julian date.
DEFFN'58(V,TO,K$,DO)	Key file access subroutine (insert/retrieve/delete keys).
DEFFN'59(V)	Set up merge array for "FIND 1ST" - file # "V".
DEFFN'60(V,K$)	Set up merge array for "FIND 1ST" = "K$" - file # "V".
DEFFN'61(V,K$,TO)	Find lowest key = "K$" in file "V".
DEFFN'62(V,U,TO)	Find next higher key in file "V".
DEFFN'63(M)	Display "M" bytes of "K$" at current row and column.
DEFFN'65(A$,E$,G$)	Display "A$" and "E$" on line 24 - Sound alarm if "G$" = "!".
DEFFN'66(V,TO)	Find lowest key in file # "V".
DEFFN'67(V,TO)	Find first physical key in file # "V".
DEFFN'68(TO)	Get next physical key from last file inquired into.
DEFFN'79(R,C,K$,L)	Display "L" bytes of "K$" at row "R"+1, column "C"+1.
DEFFN'80(P,L)	Internal subroutine - redimension S$(P)L.
DEFFN'81(Q)	Unpack basic field parameter for field number "Q".
DEFFN'82(N$)	Set "V" = file number of file named "N$" (or zero).
DEFFN'83(M,N)	Position cursor at row "M"+1, column "N"+1.
DEFFN'84(V)	Pack record in Z$() to Z$() for file # V.
DEFFN'85(V)	Unpack record in Z$() to Z$() for file # V.
DEFFN'86(V)	Build key index element, record in Z$() for file # V.
DEFFN'87(V)	Build key for unpacked record in Z$() for file # V.
DEFFN'88(V,K$(1))	Adjust key in K$(1) for ascending/descending order.
DEFFN'89(V,R$(1),DO)	Internal subroutine - record protect check/update.
DEFFN'90(N$,P)	Set device # "P" = 2 if N$="IDEAS" or "ideas".
DEFFN'91(N,DO$(N))	Set status flag N.

DEFFN'98	Error message trap - must be in user program if used.
DEFFN'99	Function key trap - must be in user program if used.
DEFFN'141(N$,K$,TO)	Same as DEFFN'41, but with file NAME.
DEFFN'142(N$,P)	Same as DEFFN'42, but with file NAME.
DEFFN'155(A,EO)	Same as DEFFN'55, but do not display resultant field.
DEFFN'159(N$)	Same as DEFFN'59, but with file NAME.
DEFFN'160(N$,K$)	Same as DEFFN'60, but with file NAME.
DEFFN'161(N$,K$,TO)	Same as DEFFN'61, but with file NAME.
DEFFN'162(N$,U,TO)	Same as DEFFN'62, but with file NAME.
DEFFN'166(N$,TO)	Same as DEFFN'66, but with file NAME.
DEFFN'167(N$,TO)	Same as DEFFN'67, but with file NAME.

DEFFN'32(N$)   Get field parameters and display screen.

Pass the name of IDEAS-developed screen mask file to this subroutine. The subroutine will then copy the contents of the work buffer (S$() 1,1008) to the record buffer (Z$()), and use all of S$() as a screen buffer. The screen mask is loaded from disk (device #3 - unless the first five characters of the screen file name are "IDEAS" or "ideas", in which case the screen mask is loaded from disk device #2), and displayed. The work buffer is then set to blanks, and the field parameters associated with the specified screen are loaded into S$() starting at byte #1009.

The variable "F" (field number) is set to zero, and the variable "FO" is set to the number of fields defined for the specified screen.

The previous contents of the work buffer are left in Z$().

The contents of the print buffer I$() are destroyed by this routine.

DEFFN'33(N$)   Get field parameters for screen N$, do not display screen.

Pass the name of an IDEAS-developed screen mask to this subroutine. The System will load the field parameters associated with the specified screen into S$(), starting at byte #1009. It will load from disk device #3, unless the first five characters of the screen file name are "IDEAS" or "ideas", in which case it will load from disk device #2.

The variable "F" (field number) is set to zero. The variable "FO" is set to the number of fields defined on the specified screen.

The contents of the work buffer S$()<1,1008> and the record buffer Z$() are not affected by this subroutine.

The contents of the print buffer I$() are destroyed by this routine.

The screen display is not changed by this subroutine.

DEFFN'34(F)   Allow input from keyboard into field number "F".

Passing a valid field number to this subroutine will cause the system to start processing the specified field according to the specifications contained in the field parameter record for that field. These are the specifications that were defined when the screen was last created or revised, and are shown on the screen documentation. The actual processing of the field varies significantly depending on the field parameters, and is explained below. One factor that remains constant, however, is that no matter how the field is processed, the contents of the field after the processing has been completed are always returned to the application program in the variables "K$" and K$(1). If the contents of the field represent a valid numeric string, the numeric representation of the field will be returned in the variable "Q", otherwise, "W" will be set to zero.

The generally recommended method of using this subroutine is shown on the next page, followed by a more detailed explanation of exactly what transpires within the subroutine itself. This is probably the most important subroutine in the entire IDEAS system, and a thorough understanding of it is necessary to properly design and define screens and application modules.

The way that this subroutine should normally be used in an application module is as follows:

```
AAAA F=F+1
BBBB GOSUB'34(F)
CCCC ON F GOTO XXXX,YYYY,ZZZZ,.......
```

Assuming that a screen file has been loaded (using DEFFN'32, for example) prior to executing line AAAA, F will be zero the first time that line AAAA is executed.

Line AAAA increments the field number, then line BBBB gets the field from the keyboard.

Line CCCC sends the program to one of several line numbers, depending on which field has just been processed and returned to the application program.

Each of these lines would perform any additional validation necessary on the field (beyond that which was specified in the field parameters), or perform any other desired functions such as to retrieve a reference record from disk, etc.

If there are no additional actions to be taken after a field has been input, other than to go on to the next field, the line number in the ON F GOTO statement corresponding to that field number would be AAAA.

If other actions are to be taken, they would be coded at the appropriate line number, then there would be a branch to line AAAA if the results were as expected, or to line BBBB to enter a new value into the same field if necessary (possibly after an error message had been displayed - GOSUB'35).

ACTUAL INTERNAL OPERATION OF DEFFN'34

When the subroutine is called, the appropriate field parameters are unpacked from the field parameter record. If there is no field parameter record for the specified field ($F > F0$), the field number is incremented until it reaches #129, which is always defined by the system as a "SPECIAL FUNCTION KEYS AND EXEC ONLY" field of 1-byte in length with no data going into the record, appearing at row 24, column 80. Assuming a valid, defined field, however, the following appears:

1. The system checks to see if there is already default field specified to provide a default value for the current field. If not, skip to Step 4.

2. If a default field were specified, and if the current field is blank, the default value is copied into the current field, and we skip to Step 4.

3. If a default field were specified, and the current field is not blank, the system checks to see if the current field is a REQUIRED field. If not, the default field is not copied. If the current field is REQUIRED and has a default field specified, the default value is copied into the current field, even if it is not blank to start with.

4. The system positions the cursor at the beginning of the field. If the field is blank, and is available for input from the keyboard, a string of boxes equal in length to the field length is displayed. If the field is not blank, and may be displayed, the contents of the field are displayed. If the field may not be displayed, and may not be input from the keyboard, a string of spaces equal in length to the field length is displayed.

5. If the field is not available for input from the keyboard, skip to step XXX.

6. The cursor returns to the beginning of the field for keyboard input. Only those characters in the specified allowable character set for the field will be permitted. If an invalid character is input, it will be ignored, the audio alarm will sound, and an error message will appear on line 24 specifying the valid character set.

   Exceptions to #6:

   a. EXEC (or RETURN) may be used at any time to terminate a field, except when:

      1) The field is a REQUIRED field and is currently blank.

      2) The field has been specified as FULL IF PRESENT and contains any blank character.

   b. BACKSPACE is valid at any time other than when the cursor is at the beginning of the field.

   c. SPECIAL FUNCTION KEYS

      1) If the SPECIAL FUNCTION KEY TRAP is ON, any special function key, at any time, will branch to DEFFN'98 in the application program, where a user-programmed test may be performed.

      2) If the SPECIAL FUNCTION KEY TRAP is OFF:

         a) If the EDIT key is touched, the field will be underscored, and the BEGIN (FN'07), END (FN'04), INSERT (FN-10), DELETE (FN'09), 5RIGHT (FN'11), 1RIGHT (FN'12), 1LEFT (FN'13), and 5LEFT (FN'14) will operate until the EDIT mode is terminated by touching EXEC, completing the entry of a field which does not require EXEC for termination, or touching the RECALL key (FN'15) which will redisplay the original contents of the field.

b) If the CANCEL key (user-specified in the "START" module creation utility) is touched at any time (other than in EDIT mode), the current program will be cancelled, any currently retrieved and protected records will be deprotected in the files, and the last MENU (or whatever program is specified in the variable "NO$") will be loaded while the load message contained in the variable "MO$" is displayed.

c) If the SKIP AHEAD KEYS are turned ON:

   i. FN'04 will cause the cursor to skip to the end of the screen.

   ii. FN'11 will cause the cursor to skip ahead 5 fields.

   iii. FN'12 will cause the cursor to skip to the next field.

   Any of these three operations will redisplay the original contents of the field in which the key was touched.

d) If the SKIP BACK KEYS are turned ON:

   i. FN'07 will cause the cursor to skip back to the first field on the screen.

   ii. FN'13 will cause the cursor to skip back to the previous field.

   iii. FN'14 will cause the cursor to skip back 5 fields.

7. If the field is available for input from the keyboard, but not displayable, the system will accept keystrokes and put the data into the field, but nine of the input data will be displayed.

8. The field may be terminated (other than as listed in #6 above) as follows:

   a. By filling the field, if EXEC is not required by the field specifications.

   b. By touching EXEC.

   c. By touching "-" as any but the first character is a NUMERIC field which permits the entry of SIGNS as valid characters. A "-" may be input as the first character in a numeric field to make the field negative, in which case the field will not be terminated until full or EXEC is touched. Or "-" may be used in place of EXEC to both terminate the field AND make the value negative. In either case, the "-" will appear at the front of the field after the field is terminated.

9. The alpha representation of the field contents will be returned to the application program in the variables "K$" and "K$(1)". If the value of the field is a valid numeric string (regardless of the field type), the numeric representation of the field's value will be returned to the application program in the variable "Q". Otherwise, the value of "Q" will be zero. The field number will be returned in the variable "F". "F" will always be the number of the file just processed. It may differ from the value of "F" that was passed to the subroutine if one of the SKIP AHEAD or SKIP BACK keys were touched.

   If the field was a "SPECIAL FUNCTIN KEYS & EXEC ONLY field type, "Q" will contain the number of the FUNCTION KEY that was used (0-31), or 32 if EXEC was used, or 240 if EDIT was used. The value of "K$" and "K$(1)" may be indeterminate in this case.

DEFFN'35(E$) Display E$ as error message on line 24, sound audio alarm.

When the subroutine is called, the words "Error Message -" will appear on line 24 followed by the contents of the variable E$ (the string passes to the subroutine), and the audio alarm (if any) will sound. The error message will remain on line 24 of the screen only until the next key is touched on the keyboard, or until another error message or operator message is displayed, whichever occurs first.

If the ERROR MESSAGE TRAP is ON, the system will branch to DEFFN'99 in the application program with the message contained in "E$" and "Error Message -" in "A$" after displaying the error message.

If the SYSTEM ERROR MESSAGES are OFF, the message will not be displayed, but the branch to DEFFN'99 will still occur if the ERROR MESSAGE TRAP is ON.

DEFFN'36 Display all fields starting at current field # "F".

This routine will display the current contents of all fields associated with the current screen before starting at field number "F" and ending with field number "FO" which have not been specified as fields which may not be displayed. "FO" is the highest field number in the screen and is set during the loading of the screen's field parameters (using DEFFN'32, DEFFN'33, or DEFFN'38). The return values will be those as described in DEFFN'37(Q) where Q=FO.

"F" is normally the number of the current field to be processed. The value of "F" is not changed by executing this subroutine.

"If "F" is zero, the first field to be displayed will be field #1.

A fatal error may be caused by calling this subroutine when the value of "F" is greater than the value of "FO".

DEFFN'37(Q) Display the contents of the field number "Q".

This subroutine will display the contents of the field number specified in the subroutine call unless the field was specified as one which does not permit display. The contents of the field will be returned in the variables "K$" and "K$(1). If the field contains a valid numeric value, it will be returned in the valiable "Q", otherwise "Q" will be zero.

A fatal error may be encountered if the value passes to the subroutine is less than 1 or greater than FO.

DEFFN'38(N$) Get field parameters, display screen, keep current record.

This subroutine operates in exactly the same manner as DEFFN'32, except that the contents of the work buffer (S$() <1,1008 >) which were copied into the record buffer Z$*() are returned to the work buffer upon completion of the subroutine, thus the current record is preserved. (It is also left in Z$().) This subroutine is especially useful when 2 or more screens are required to generate a single record.

As in DEFFN'32, the print buffer I$() is destroyed by the user of this subroutine.

DEFFN'39(N$,P) Get limits of file "N$" on device # "P".

Pass a file name (either a program file or a data file), and a disk device number to this subroutine. The system will then get the limits of the specified file from the specified disk unless the first 5 characters of the file name are "IDEAS" or "ideas" in which case the vlue of "p" will be changed to 2.

The limits are returned to the application program in the following variables:

N$	File name
P	Device number used
T	Device number used
A	Starting sector of file
M	Ending sector of file
N	Number of sectors used in the file

A fatal error will be encountered if the specified file does not exist.

DEFFN'40(N$,E$) Load program "N$", displaying load message "E$".

This subroutine first checks the key buffer F$() for any keys which may exist there. The existence of a key in the key buffer indicates to the system that a record has been read AND PROTECTED on disk. If there are any keys in the key buffer, the system re-reads the indicated records, and re-writes each of them with the protect byte turned off. While it is doing this, it displays the message "Cleaning up data files" in the center of a blank screen.

After the file has been cleaned up, or immediately if the key buffer is blank, the system displays "Loading ..." followed by the specified load message in the center of a blank screen and loads the program specified in "N$" form device #4 starting at line 1000. (If the 1st 5 characters of the program name are "IDEAS" or "ideas", the program will be loaded from device #2.)

A fatal error will occur if the specified program does not exist, and may occur if the specified program does not begin with line 1000.

DEFFN'41(V,K$,TO) "GET" and unpack record from file number "V".

The following variables are passed to this subroutine:

V	The file number of the primary or alternate key file
K$	The key associated with the desired record
TO	Record protect flag AND position within the work buffer

This subroutine searches the specified file for the given key, reads the record, unpacks it and places the record in the record buffer Z$() starting at byte #1. If absolute value of TO is 1 or greater, the record is also copied into the work buffer S$() starting at byte # ABS(TO).

If the value of TO is negative, the record will not be protected, and the keys will not be written into the key buffer.

If the value of TO is zero or positive, the record will be protected with the terminal number of the requesting terminal written to disk in the protect byte of the record in binary, and the primary and any associated alternate keys will be written into appropriate elements of the key buffer.

If the record was found unprotected, the above will be true and the value of "Q" returned to the application program will be positive.

If the record was found protected by another terminal, it will still be read and unpacked to the record buffer, and transferred to the work buffer if specified. However, the keys will not be written into the key buffer, nor will the record be protected by the current terminal. An error message will be displayed stating that the record is currently in use and specifying the protecting terminal. The value of "Q" will be negative.

If the record was not found, the value of "Q" will be zero and an error message will be displayed stating that the record is not on file.

The value of "K$" as passed to the subroutine must be already adjusted using DEFFN'88 if any part of it is specified to be in DESCENDING sort order.

A fatal error will occur if "V" is less than 1, or greater than the number of files currently open.

A fatal error will occur if the absolute value to TO plus the unpacked record record length exceeds 1009.

DEFFN'42(V,P)   Pack record and PUT it into file number V.

Pass this subroutine a file number (V) and a position in the work buffer (P). The system will copy the record from the work buffer starting at byte P into the record buffer (unless P=0 in which case the record is already assumed to be in the record buffer Z$().)

The system then checks the primary and alternate keys (if any) for that record against any that may exist in the key buffer for the primary and any alternate files associated with the given file number. If there is a key in the key buffer then the "PUT" operation is an update, because there must have just been a "GET" to cause those key buffer positions to be filled.

If an update is sensed, each key in the record is compared to the corresponding key in the key buffer and if they are unequal, the old one is deleted from the file and the new one is inserted.

If an operation is only a "save" and not an update, then each key in the record is inserted into the appropriate key file.

The key buffer elements associated with the given file are then cleared, the record is packed, and it is saved in the primary file with the protect byte turned off.

If the record was stored successfully, a positive value will be returned to the application program in the variable "Q". If the record was not stored successfully, an error message will be displayed giving the reason (i.e., File full, or Illegal duplicate key) and specifying the file name in which the problem occured. Any keys which may have been updated prior to the occurance of the error condition will have been restored to their original condition.

DEFFN'44(R,C,K$L)   Display L bytes of K$ at row R, column C.

Given a row and column on the screen where an alpha string is to be displayed, the string of characters, and the number of characters in the string to be displayed, this subroutine displays the specified number of characters in the given string starting at the specified row and column.

R	=	Row on the screen for the first character in the string.
C	=	Column on the screen for the first character in the string.
K$	=	The string of characters to be displayed (max=64).
L	=	The number of characters to be displayed. If L=0, the number of characters displayed will be equal to the length of K$ (LEN(K$)).

Note:

IDEAS counts rows and columns starting at 1 i.e., rows 1-24, columns 1-80, rather than 0-23, 0-79.

If the string represents a valid numeric value, the numeric value will be returned to the application program in the variable "Q". Otherwise, "Q" will be zero.

DEFFN'45(Q) Retrieve the contents of field number "Q".

Passing a valid field number to this subroutine will cause the contents of the specified field to be returned to the application program in the variables "K$" and "K$(1)". If the field represents a valid numeric value, the number value os the field will be returned in the variable "Q". Otherwise, "O" will be zero.

A positive field number will cause the field to be retrieved from the work buffer S$() < 1,1008 >. A negative field number will cause the field to be retrieved from the record buffer Z$(). If you are retrieving from the record buffer, be certain that the contents of the record are as desired and have not been changed by another "PUT" or "GET" operation.

Unpredictable and/or fatal results may occur if the absolute value of the field number is zero or greater than FO (the maximum number of fields in the current screen).

DEFFN'46(Q,K$) Put the contents of K$ into field number "Q".

Pass this subroutine a field number and an alpha string. The number of characters in the string equal to the length of the specified field will be copied into the field.

If the field number is positive, the field will then be displayed on the screen.

If the field number is negative, the field will not be displayed.

The numeric representation of the string will be returned to the application in the variable "Q" if the string were a valid numeric string. Otherwise, "O" will be zero.

Unpredictable and/or fatal results will occur if the absolute value of the field number is zero or greater than FO.

DEFFN'47(P,K$,L) Copy "L" bytes to/from "K$" from/to record position "P".

This subroutine allows you to copy a string of up to 64 characters from K$ to either the work buffer S$() 1,1008 or the record buffer Z$(), or to copy a string of up to 64 character to K$ either from the work buffer or the record buffer.

P	=	Position in the work or record buffer for copy to start. If P > 0, copy the ABS(L) bytes of K$ to the work or record buffer starting at byte number SBS(P). If P < 0, copy the ABS(L) bytes from the work or record buffer starting at byte # ABS(P) to K$.
L	=	Length of string to be copied. If L > 0, copy to or from the work buffer S$() <1,1008>. If L < 0, copy to or from the record buffer Z$().

The system returns the copied string in the variables "K$" and "K$(1)". If the string is a valid numeric string, the numeric value is returned in the variable "Q", otherwise, "Q" is zero.

DEFFN'48(C,K$(1),L)   Copy"L" bytes of "K$(1)" to print buffer @ position "C".

The print buffer is I$() which consists of an array of 128 elements of 2-bytes each. When the print buffer is initialized, the first 254 bytes are set to spaces and the last 2 bytes are set to HEX(0000). Byte 255 is used as a part line length indicator.

Use of DEFFN'48 assumes that the print buffer has not been destroyed since it was last initialized (DEFFN'50), printed (DEFFN'49), or updates (DEFFN'48). Subroutines which destroy the print buffer are as follows:

1. The program loading subroutine (DEFFN'40).

2. The screen and/or field parameter loading subroutine (DEFFN'32, '33, '38.

3. The record packing or unpacking subroutines (DEFFN'84, 85).

4. The key assembly subroutines (DEFFN'86, '87, '88).

5. The file access subroutines (DEFFN'41, '42, '58, '59, '60, '61, '62, '66, '67, '68, '89', '141, '142, '159, '160, '161, '162, '166, '167).

The variables passed to this subroutine are as follows:

C   The position where the beginning of the specified string is to be placed in the print buffer.

If C > 0	The string will be copied into the print buffer starting at byte # C.
If C = 0	The string will be copied into the print buffer starting at the next available byte (current line length + 1).
If C < 0	The string will be copied into the print buffer starting at the next available byte + the absolute value of "C".

K$(1)   The string which is to be copied into the print buffer.

L   The number of bytes of K$(1) which are to be copied into the print buffer.

If L > 0	The specified number of bytes will be copied.
If L = 0	The number of bytes copied = LEN(K$(1))

In any of the above cases, the line length (byte 255) will be updated to reflect the new line length.

A fatal error will be encountered if you attempt to copy beyond byte 254.

DEFFN'49(L) Print "L" bytes of print buffer, reset buffer.

Using the printer at device address #1, print the contents of the print buffer starting at byte 1 for "L" bytes, unless "L" = zero, in which case use the line length in byte 255 of the print buffer as the length to be printed. Then drop through to DEFFN'50 to reset the print buffer.

This subroutine may be used immediately after a DEFFN'50 to generate a line feed on the printer.

DEFFN'50 Reset Print buffer.

This subroutine resets the printer buffer in preparation for the use of DEFFN'48 to copy text or data into the print buffer. It sets the first 254 bytes of I$() to spaces, and sets the line length (Byte 255) to zero (HEX(00)).

DEFFN'51(Q,M) Copy field # "Q" to screen buffer @ screen column + "M".

This subroutine copies the contents of the specified field from the work buffer (S$()) to the print buffer, starting at the position in the print buffer equal to the column on the screen where the field starts plus the value of the variable M (or - if $M < 0$). It updates the line length byte in the print buffer.

DEFFN'52(Q) Unpack all field parameters for field number "Q".

Unpacks the field parameters for the specified field from the field parameters record for that field as follows:

T	Field type (1 - 9) 1 Digits only 2 Digits & signs 3 Digits & decimal point 4 Any numeric 5 Uppercase letters 6 Uppercase letters & digits 7 Uppercase letters, numerics, & punctuation 8 Any character 9 Special functions keys, EDIT & EXEC only
D	Number of decimal places
P	Position in the record (or in the work buffer)
L	Field Length
R	Row on screen of first byte in field (-1)
C	Column on screen of first byte in field (-1)

DO	Keyboard/Display enable/disable flag
	DO=0 Keyboard enabled, display enabled
	DO=1 Keyboard disabled, display enabled
	DO=2 Keyboard enabled, display disabled
	DO=3 Keyboard disabled, display disabled
TO	Field termination enable flag
	TO=1 Optional field, need not be filled, terminate on full or EXEC
	TO=2 Optional field, need not be filled, EXEC required to terminate
	TO=3 Optional field, must be full if used, terminate on full or EXEC
	TO=4 Optional field, must be full if used, EXEC required to terminate
	TO=5 Required field, need not be filled, terminate on full or EXEC
	TO=6 Required field, need not be filled, EXEC required to terminate
	TO=7 Required field, must be full if used, terminate on full or EXEC
	TO=8 Required field, must be full if used, EXEC required to terminate
W$	Justification & fill character indicator

W$=HEX(00)	Left justified - no fill character
W$=HEX(01)	Left justified - no fill character
W$=HEX(02)	Right justified - space fill unused high order positions
W$=HEX(03)	Right justified - zero fill unused high order positions

A fatal error may be encountered if the value of "Q" is greater than 129, less than -129, or zero.

DEFFN'53(E$) Display message to operate on line 24.

The word "Attention: ", followed by the string passed to the subroutine (up to 64 characters) will be displayed on line 24 and will remain there until the next key is touched. If the ERROR MESSAGE TRAP is ON, this will then branch to DEFFN'99 in the application program, where checking for "Attention: " in A$ will allow you to return.

DEFFN'54(P,K$(1)) Internal routine used in data validation routines.

This subroutine has no viable use in an application program. It is strictly an internal subroutine used in the date utility section of the system subroutine module.

DEFFN'55(A,EO) Round value "EO" to field spec. dec., put in field # "A".

This subroutine will round the value passed to it as EO to the number of decimal places specified in the field parameter record for field number "A", convert the value to an alpha string with justification and fill character as specified by the field parameters, place the string in field number "A" in the work buffer and display it if the field parameters allow the field to be displayed.

If the value converts to a string that exceeds the field length, only the most significant digits that fit into the field will be used, and an error message will be displayed showing the actual value stating that the value does not fit in the specified field.

The alpha representation of the field is returned to the application program in the variables "K$" and "K$(1)". The numeric value is returned in the variable "Q".

A fatal may occur if the specified field does not exist or if its length exceeds 13 characters.

DEFFN'56(K$,Q)   Date validation/conversion subroutine

This subroutine allows a date to be input as an alpha string in any of 4 formats. It checks the validity of the date, and if valid, provides a total of 6 formats of the same date to the application program. If the date is invalid, an error message is displayed showing the invalid date and stating that it is invalid.

The date is passed to the subroutine as in alpha string and a number indicating the format of the date is passed as a numeric value (1,2,3, or 6). Ex. August 29, 1979.

1.	MMDDYY format	GOSUB'56("082979",1)
2.	DDMMYY format	GOSUB'56("290879",2)
3.	YYMMDD format	GOSUB'56("790829",3)
6.	YYDDD (Julian) format	GOSUB'56("79241",6)

The results will be returned to the application program in the D$() for a valid date as follows:

1.	MMDDYY format	D$(1) = 082979
2.	DDMMYY format	D$(2) = 290879
3.	YYMMDD format	D$(3) = 790829
4.	MMM DD YY format	D$(4) = AUG 29 79
5.	DD MMM YY format	D$(5) = 29 AUG 79
6.	YYDDD (Julian) format	D$(6) = 79241

In addition, the Julian date will also be returned in the variable "Q". If the date were invalid, the value of "Q" is zero.

A fatal error may occur if a format number other than 1, 2, 3, or 6 is used, or if the string does not contain 6 digits (5 for Julian).

DEFFN'57(L)   Convert Julian date.

Passing a Julian date in the form YYDDD as a numeric value to this subroutine will return the date to the application program in the same manner as DEFFN'56 (qv).

DEFFN'58(V,TO,K$,DO) Key file access subroutine (insert/retrieve/delete).

V	=	File number of key file to be accessed
TO	=	Protect flag: TO=0 - do not protect record, TO > 0 - protect
K$	=	The key to be inserted, retrieved, or deleted
DO	=	Operation Code
		DO = 0 Retrieve key K$
		DO = 1 Insert key K$, even if it is a duplicate key
		DO = 2 Insert key K$, it it is not a duplicate key
		DO = 3 Delete key K$

If the specified operation is NOT successful, a value of zero is returned to the application program in the variable "O". If the record accessed is currently in use by another terminal, the value of "Q" will be the negative of the terminal number of the "using" terminal.

If the operation was successful, the value of "O" will be positive, the disk device number of the platter containing the record will be "T", the sector number where the record starts will be "E", and the byte number within that sector where the protect byte for that record (the byte immediately preceeding the PACKED record) will be "C". The pointer associated with the record will be found in the 3-byte variable R$(1).

DEFFN'59(V) Set up merge array for "FIND 1ST" - file # "V"

This subroutine reads the first key and pointer from each bucket in the files and places it in ascending sort order in the array U$(). it does NOT actually retrieve any records. It merely sets up the merge array so that the application may process the file by using FIND NEXT (DEFFN'62) for each record, rather than having to perform a FIND FIRST (DEFFN'66) followed by a series of FIND NEXTs.

Each element in U$() consists of string variable K+6 bytes where K = the length of the actual key. The 3 bytes immediately following the actual key represent the pointer associated with that key, and the last 3 bytes are the sector number and byte number from which the key was taken.

DEFFN'60(V,K$) Set up merge array for "FIND 1ST > =K$" - file # "V"

This subroutine performs the same function as DEFFN'59 except that you may start at other than the lowest key in the file. It will set up the merge array such that you may begin processing with the lowest key that is equal to or greater than the value passed to the subroutine.

DEFFN'61(V,K$,TO)   Find lowest key > =K$ in file # "V" (FIND 1ST > =K$)

Sets up the merge array as in DEFFN'60, and retrieves the record corresponding to the lowest key in the merge array. If TO < = 0, the record is not protected. If TO > 0, the record is protected. The record is unpacked and left in the record buffer Z$(). If INT(ABS(TO)) <> 0, the record is also copied into the work buffer S$() < 1,1008 > starting at the byte indicated by the absolute value of TO. If Q < 0, the record was found not protected. If Q > 0, the record was found protected. If Q=0, a record was not found (end of file).

DEFFN'62(V,U,TO)   Find next higher key in file number "V".

Use of this file assums the prior use of DEFFN'59, DEFFN'60, or DEFFN'61, for the same file number. Otherwise a fatal error may occur. Sequential processing operations may NOT be used on more than 1 file at a time.

V	=	File number
U	=	Previous key deletion flag - for most normal sequential processing operations, the value of "U" should always be 1. However, if the file is being processed sequentially for the purpose of deleting a certain record, the value of "U" must be zero on the next use of DEFFN'62 following the deletion of the record immediately preceeding the same files.
TO	=	Protect flag/position indicatior.

TO < =-1	The record will NOT be protected, it will be unpacked to the record buffer Z$() and copied to the work buffer S$() 1,1008 starting at byte # ABS(TO).
-1 < TO <=1	The record will NOT be protected, and will only be unpacked to the record buffer Z$().
0 < TO < 1	The record will be protected and will be unpacked to the record buffer Z$() only.
TO > =1	The record will be protected and will be unpacked to the record buffer Z$() and copied to the work buffer S$() 1,1008 starting at byte #TO.

The return value of "Q" are as follows:

AQ < 0	The record is currently in use on another terminal.
Q=0	The end of the file has been reached - a record was not found.
Q > 0	An unprotected record was found.

DEFFN'63(M)   Display "M" bytes of K$ at the current row & column.

This subroutine will display the first "M" bytes of K$ of the current contents of the alpha variable "K$" starting at the current cursor position.

DEFFN'65(A$,E$,G$)   Display "A$" and "E$" on line 24, sound alarm if G$="!".

This subroutine gives the programmer the facility to create a message or error message of up to 80 characters to be displayed on line 24 of the screen and left on the screen until the next key is touched. The first 16 characters of the message go into the variable "A$" and the remainder go into "E$". If the character "!" is passed to "G$", the audio alarm (if any) will sound. If any other character is used, it will not.

If the ERROR MESSAGE TRAP is ON, this will branch to DEFFN'99 in the user program. If you do not wish this to occur, you may first turn off the error message trap by using DEFFN'91.

DEFFN'66(V,TO)   Find lowest key in file number "V".

This operates in exactly the same way as DEFFN'61 except that you do not pass a specific key value to the subroutine. Instead, it finds the lowest logical key in the file. The protect/position code TO and the return value Q are the same as in DEFFN'61 or DEFFN'62.

DEFFN'67(V,TO)   Find first physical key in file number "V".

Instead of finding the lowest logical key in a file, this subroutine finds the first PHYSICAL key in bucket number zero (the first bucket) in the file. It does NOT use the merge array, so it may be used in conjunction with sequential processing operations on another file if necessary.

The protect/position code TO, and the return code Q are the same as in DEFFN'61.

DEFFN'68(TO)   Find next physical key.

Instead of finding the next logical key as in DEFFN'62, this subroutine finds the next PHYSICAL key IN WHATEVER FILE WAS LAST ACCESSED WITH A RETRIEVAL OR DELETION SUBROUTINE.

Please note that a file number is not specified.

This routine uses the file that was last accessed for anything but a "PUT" or a GOSUB'58.

The value of TO and the return code Q are the same as in DEFFN'62.

This subroutine, combind with DEFFN'67, is useful for a quick scan of a file when key sequence processing is not necessary.

DEFFN'79(R,C,K$,L) Display L bytes of K$ at row R+1, column C+1

This subroutine operates in exactly the same way as DEFFN'44 with the following exception:

IDEAS normally uses row and column designations starting at the number 1.

This subroutine may be used for row and column designations starting with 0, i.e., GOSUB'79(3,0,"ABCDEFGHIJ",5) would cause ABCDE to be displayed at row 4, column 1.

DEFFN'80(P,L) Internal subroutine - redimension S$(P)L.

This subroutine should be of little or no use to the programmer. It is used in several places internally in the IDEAS system subroutine module to redimension S$().

USE OF THIS SUBROUTINE WITHOUT A COMPLETE UNDERSTANDING OF THE INTERNAL LOGIC OF IDEAS AND WITHOUT RETURNING THE DIMENSIONS OF S$() TO THEIR APPROPRIATE VALUES WILL UNDOUBTEDLY CAUSE FATAL ERRORS!

DEFFN'81(Q) Unpack basic field parameters in field # Q.

This subroutine performs the same function as DEFFN'52 except as follows:

1. W$ is not set.
2. DO is not set.
3. R is not set - R will be 4*R + DO (as compared to the results of DEFFN'52).

DEFFN'82(N$) Set "V" = file number of file named by "N$".

Passing the name of a currently open primary or alternate file to this subroutine will set the variable "V" equal to the file number for that file. If the specified file does not exist (has not opened), the value of "V" will be zero.

DEFFN'83(M,N) Position cursor at row M, column N.

As in DEFFN'79, this subroutine assumes row and column numbers starting at 0. GOBUB'83(3,0) will position the cursor at row 4, column 1.

DEFFN'84(V)   Pack record in Z$() to Z$().

This subroutine packs the record in the record buffer Z$() according to the record description parameters loaded when the file was opened. Any specified numeric fields at the beginning of the record are packed 2:1, and any uppercase alpha fields immediately following are packed 4:3. The resulting packed records is left in the record buffer Z$().

DEFFN'85(V)   Unpack record in Z$() to Z$() for file # V.

This subroutine assumes that a packed record from file # V is in the record buffer Z$(). It unpacks the record according to the record description parameters loaded when the file was opened and leaves the unpacked record in the record buffer Z$().

DEFFN'86(V)   Build key index element, record in Z$() for file # V.

This subroutine currently performs the same function as DEFFN'87. It may be used for additional purposes should the file types supported by IDEAS be expanded at a later date.

DEFFN'87(V)   Build key for unpacked record in Z$() or S$().

If V > 0, the record is assumed to be in the record buffer, Z$(). If V < 0, the record is assumed to be in the work buffer, S$() < 1,1008 >, starting at byte #1.

This subroutine builds the key for specified file (ABS(V)) consisting of up to 3 fields. It complements any field specified to be in descending sort order.

The correct, complemented key is left in the variable "K$(1)". The uncomplemented key is left in the variable "K$".

DEFFN'88(V)   Adjust key in K$() for ascending/descending sort order.

A key for file # "V" is assumed to be in the variable "K$(1)". This subroutine adjusts any necessary component of the key to reflect a specified descending sort order for the particular field component. The complemented key is left in K$(1), and the original uncomplemented key is left in K$.

DEFFN'89(V,R$(1),DO) Internal subroutine - record protect check/update.

V	=	File number
R$(1)	=	Pointer to record (found in key index)
DO	=	Operation code

Given the file number pointer and operation code, this subroutine will read the protect byte for a given record and set the value of "Q" to the negative of the protecting terminal number if the record is protected.

T	=	The device number of the disk on which the record resides.
E	=	The sector number on disk # T where the record begins.
C	=	The byte number within sector # E where the protect byte for the specified record exists. This byte is the byte immediately preceeding the packed record.

If the value of DO is zero, AND IF THE VALUE OF TO IS POSITIVE, the record will be protected by the terminal using this subroutine.

DEFFN'90(N$,P) Set device # P = 2 if N$ = "IDEAS" or "ideas".

This subroutine is used internally in all of the IDEAS program and/or data file loading subroutines. A program or data file name and a disk device number is passed to the subroutine. If the first 5 characters in the file name are either "IDEAS" or "ideas", the device number will be set to 2. Otherwise, the device number will be left unchanged. In either case, the device number is returned to the application program in both "P" and "T".

DEFFN'91(N,OO$,(N)) Set status flag "N".

This array OO$() consists of 9 bytes of flags as follows:

OO$(1)	The binary value of special function key number used as the CANCEL key. If this is set to a function key number which is not available on the keyboard, the CANCEL function will be disabled.
OO$(2)	Skip-ahead key enable flag. If this flag is set to "Y", the skip-ahead keys are turned ON. Otherwise, they are not available to the opeator.
OO$(3)	Skip-back key enable flag. If this flag is set to "Y", the skip-back keys are turned ON. Otherwise, they are not available to the operator.

00$(4)	Error message flag. If this flag is NOT set to "N", the error messages will be displayed as usual. If this is set to "N", no error messages will be displayed.
00$(5)	Error message trap flag. If this flag is NOT set to "Y", the error message trap will be inoperative. If this flag is set to "Y", any error message (DEFFN'35), operator message (DEFFN'53), or general message (DEFFN'65) will cause a branch to the error message trap subroutine (DEFFN'99) in the application program.
00$(6)	Special function key trap. if this flag is NOT set to "Y", the special function key trap will be inoperative. If it is set to "Y", then ANY special function key tocuhed at any time (including the CANCEL key or a function key used in a FUNCTION KEYS & EXEC field type) will cause a branch to the special function key trap subroutine (DEFFN'98) in the application program.
00$(7)	Reserved for future use.
00$(8)	The binary representation of the terminal ID number.
00$(9)	The CPU type identification flag. = T if the CPU is a 2200T. = V if the CPU is a 2200VP. = M if the CPU is a 2200MVP.

Any of these flage may be reset by the use of DEFFN'91 by passing the element number (1-9) and the new flag value to the subroutine.

DEFFN'98 — Special functiuon key trap: must be in application if used.

This is a subroutine that is NOT contained in the system subroutine module. If it is desired, it must be coded by the application programmer in each application program where it is needed. If the special function key trap is turned on, any function key will cause a branch to this subroutine where the application program may make a test on the function key number (contained in the variable G$(6) as a binary value) and perform any desired function accordingly.

If no programmed function is performed, a RETURN will suffice to return control to the point in the program execution where the function key was touched. Otherwise, 2 RETURN CLEAR's should be used to clear the subroutine stack if you do not return to that point.

DEFFN'99 Error message trap: must be in application if used.

This is a subroutine that is NOT contained in the system subroutine module. If it is desired, it must be coded by the application programmer in each application program where it is desired. If the special function key trap is turned on, any error message (DEFFN'35), operator message (DEFFN'53) or general message (DEFFN'65) will cause a branch to this subroutine.

Error messages and operator messages may be identified by the contents of the variable A$ ("Error message -", and "Attention:" respectively). Also, the contents of E$ will be the actual message.

Any action taken in this subroutine (except loading another program) should be terminated by a RETURN to the calling point in the main program execution because the original error message may have been called in any of several levels of subroutines' and there is a distinct danger of causing an overflow of the subroutine stack if this option is not used carefully.

DEFFN'141(N$,K$,TO)

This subroutine operates in exactly the same way as DEFFN'41(V,K$,TO) except that you supply a file name N$ instead of a file number V.

DEFFN'142(N$,P)

This subroutine operates in exactly the same way as DEFFN'42(V,P) except that you supply a file name N$ instead of a file number V.

DEFFN'155(A,EO)

This subroutine operates in exactly the same way as DEFFN'55(A,EO) except that the field is not displayed and is always assumed to be right-justified, space-filled.

DEFFN'159(N$)

This subroutine operates in exactly the same way as DEFFN'59(V) except that you supply a file name N$ instead of a file number V.

DEFFN'160(N$,K$)

This subroutine operates in exactly the same way as DEFFN'60(V,K$) except that you supply a file name N$ instead of a file number V.

DEFFN'161(N$,K$,TO)

This subroutine operates in exactly the same way as DEFFN'61(V,K$,TO) except that you supply a file name N$ instead of a file number V.

DEFFN'161(N$,U,TO)

This subroutine operates in exactly the same way as DEFFN'61(V,U,TO) except that you supply a file name N$ instead of a file number V.

DEFFN'162(N$,U,TO)

This subroutine operates in exactly the same way as DEFFN'62(V,U,TO) except that you supply a file name N$ instead of a file number V.

DEFFN'166(N$,TO)

This subroutine operates in exactly the same way as DEFFN'66(V,TO) except that you supply a file name N$ instead of a file number V.

DEFFN'167(N$,TO)

This subroutine operates in exactly the same way as DEFFN'67(V,TO) except that you supply a file name N$ instead of a file number V.

NOTE:

In all of the above (except DEFFN'55), the use of non-existent file name will cause a fatal error to occur.

APPENDIX B
DATA RECORD DEFINITION FILES

I. Relative sector 0 (loaded into I$() in BA mode)

POSITION STARTING BYTE #	LENGTH	DESCRIPTION
1	8	Logical file name
9	3	Vol 1 disk address
12	8	Primary file name
20	1	Number of volumes
21	1	Record blocking option #
22	1	Performance option #
23	2	File type
25	8	Key 1 name
33	1	Key 1 order
34	3	Key 1 position in record
37	1	-not used-
38	2	Key 1 length
40	1	Key 1 type
41	8	Key 2 name
49	1	Key 2 order
50	3	Key 2 position in record
53	1	-not used-
54	2	Key 2 length
56	1	Key 2 type
57	8	Key 3 name
65	1	Key 3 order
66	3	key 3 position in record
69	1	-not used-
70	2	Key 3 length
72	1	Key 3 type
73	2	Key length
75	3	# of fields
78	4	Unpacked length
82	4	# of packed numerics
86	4	# of packed alphas
90	4	Packed length
94	3	Vol 2 disk address
97	3	Vol 3 disk address

(Continued)

POSITION STARTING BYTE #	LENGTH	DESCRIPTION
100	3	Vol 4 disk address
103	3	Vol 5 disk address
106	3	Vol 6 disk address
109	3	Vol 7 disk address
112	3	Vol 8 disk address
115	6	Last revised date
121	8	Physical file name
129	8	Alternate key file #1
137	8	Alternate key file #2
145	8	Alternate key file #3
153	8	Alternate key file #4
161	8	Alternate key file #5
169	8	Alternate key file #6
177	8	Alternate key file #7
185	8	Alternate key file #8
193	8	Alternate key file #9
201	8	Alternate key file #10
209	8	Alternate key file #11
217	8	Alternate key file #12
225	8	Alternate key file #13
233	8	Alternate key file #14
241	8	Alternate key file #15
249	8	Alternate key file #16

II. Relative sectors 1-6 contains field names, field type, position of field in record and field length. These sectors are loaded into Y1$() in DA mode. Y1$ is dimensioned to Y1$(24)62. After the sectors are loaded into Y$(), the system does a MATCOPY to Y$(). Y$() is dimensioned to Y$(128)11.

The first three bytes of each element of Y$() is the result of a $PACK. These three bytes contain the following information: 1) field type, 2) position of field in record, and 3) field length.

To retrieve the information in these 3 bytes, do the following.
Assume P4$-STR(Y$(I),1,3) where I is any field from 1 to 128.

```
UNPACK (######)P4$ to Q
CONVERT Q TO K$, (######)
K$(1)=HEX(100110031002)
$UNPACK(F=K$(1))K$ to M1, M2, M3
```

Where M1 is the field type, M2 is the position within the record and M3 is the length of the record.

DATA FILE (Relative sector 0 on each Volume of Data File)
P1$32

Position in E$()	Position in P1$	Description
12	1	File Type
13	2	Key Length
14	3	Giskie Length
15	4	Fiskie Length
16	5	HASH Length
17	6	# Fis/Bucket
18	7	# Rec/Block
19	8	Key 1 Length
20	9	Key 2 Length
21	10	Key 3 Length
22	11	Key 1 Pos
24	13	Key 2 Pos
26	15	Key 3 Pos
28	17	# of Buckets
30	19	Vol 1 Bucket Length
32	21	# Sectors/Record Length
34	23	# Records/Bucket
36	25	Uncompressed Record Length
38	27	# of Compressed Num
40	29	# of Compressed Alpha
42	31	Compressed Length

Other values of E$()

1-8	Filename
9	File type
10	Disk Dev #
11	Disk Address (last digits)

Hashing Algorithm is in DEFFN'58 (line 356 in ID-SUB-M and ID-SUB-V)

The following variables are used in the code generated by IDEAS and should not be used for other purposes by the programmer:

A	file type
C	key length
D	gross index sector
B	
D	home bucket number
EO	
L	fine index sector
M	hash length
N	number of fine index sectors per bucket
Q	total number of buckets
F	field number
FO	number of fields
S$()	work buffer
Z$()	work buffer
I$()	work and print buffer
H$()	work buffer
K$	
K$()	value of field is in K$ and K$()
Q	numeric value of field if any, otherwise zero
R$()	pointer to data record
@T$()	
T$()	translation table for numeric pack and unpack
@TO$()	
TO$()	list of valid characters
U$()	merge array for FIND FIRST, FIND NEXT
@W$()	set of pointers to the !TO$() -- used in validation
W$()	
E$()	45 bytes in length

FIELD PARAMETERS

BYTE 1	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6	BYTE 7	BYTE 8
A B	C		D	E	F	G	H

A = Field type (1-9) (Valid input character set)

1	=	Digits
2	=	Digits and signs
3	=	Digits and decimal point
4	=	Digits, signs, and decimal point
5	=	Uppercase letters
6	=	Uppercase letters and digits
7	=	Uppercase letters, digits, signs, and punctuation
8	=	Any character
9	=	Function keys, EDIT, and EXEC

B = Number of decimal places (valid for types 1-4)

C = Starting position of field in work buffer

D = Length of field

E = Row (on screen) and entry and display enable flags externally, (to the programmer creating the screen). The rows are numbers 1-24. Internally, they are 0 to 23. In the field parameter, the number is authorized by a (0 to 92).

The row for the field, then may be found by the following formula:

R = INT(E/4)+1

By storing the row as described above, the lowest 2 bits of byte 5 in the field parameter are available for flags. The "one" bit is used as the display enable/disable flag, and the "two" bit is used as the keyboard input enable/disable flag. In either case, a zero in the appropriate bit enables the function.

2-BIT	1-BIT	COMBINED VALUE	DISPLAY	KEYBOARD INPUT
0	0	0	Enabled	Enabled
0	1	1	Disabled	Enabled
1	0	2	Enabled	Disabled
1	1	3	Disabled	Disabled

F = Column (on screen). Externally, the column numbers are 1-80. Internally, they are 0-79 in the field parameters.

G = Miscellaneous Field Parameters. This byte contains 5 miscellaneous field parameters, 3 in the high order half-byte, and

HIGH ORDER				LOW ORDER			
G_8	G_7	G_6	G_5	G_4	G_3	G_2	G_1

G8	Not used	
G7	1 = Required field	0 = Optional field
G6	1 = Must be filled if used	0 = Need not be full
G5	1 = EXEC required	0 = Terminate when full
G4	Not used	
G3	Not used	
*G2	1 = Zero fill	0 = Space fill
G1	1 = Right justify	0 = Left Justify

*G2 is not applicable if field is left justified.

H = Default value field number. This is the only byte in the field parameter which is stored in full binary location. It indicates the field number in the current or previous record which is to supply the default value (if any) for the current field.

If H=HEX(7F) (decimal 127), there is not default value.

If H HEX(7F), the default value is supplied by the contents of field #(127-VAL(H)) in the current record.

If H HEX(7F), the default value is supplied by the contents of field # (VAL(H)-127 in the previous record.

H(in HEX)	DEFAULT FIELD #	RECORD
00	127	Current
01	126	Current
.		
.		
.		
7E	1	Current
7F	NONE	
80	1	Previous
81	2	Previous
.		
.		
.		
.		
FE	127	Previous
FF	128	Previous

APPENDIX C
PARTIAL VARIABLE LISTING

IDEAS - Partial Variable Listing

Dimension		Variable	Description
		F	Field Numbers
		FO	Total # of Fields
		FO$	HIKAM File Name
S$(255)8		S$()	Screen Buffer
Z$(128)8		Z$()	Work Buffer
I$(128)2		I$()	Work & Print Buffer
		H$()	Work Buffer
T$(2)34		T$()	Translation Table for Numeric Pack & Unpack
TO$(2)45		TO$()	List of Valid Characters
		U$()	Merge Array for Find First, FINDNEXT (Dimensioned When Creating Start Module)
		UO	Number of Opened Files
WO$(9,2)1		WO$()	Set of Pointers to TO$() (Used in Validation)
	*	X$()	Contains Valid Disk Selections
Y1$(24)62	*	Y1$()	Load Variable for Field Names
Y$(128)11	*	Y$()	Search Variable for Field Names
J$(1)2		J$()	Search to Variable in MAT Search
G$(10)1		G$()	GIO ARG 2 Variable
G$10		G$	GIO ARG 2 Variable
D$(6)9		D$()	Date Storage Variable
		D$	Numeric Form of Date
M$(12)5		M$()	Month Storage Variable
F$16		F$	Variable Used in $PACK & $UNPACK
OO$(9)1		OO$()	Variable for Status Flags
		E$()	File Parameters Buffer (Dimensioned When Creating Start Module)
		F$()	Key Buffer (Dimensioned When Creating Start Module)

* Not used in applications.

APPENDIX D
IDEAS SYSTEM UTILITIES

I. START-UP PROGRAMS

PROGRAM	DESCRIPTION
1. IDEAS	IDEAS package disk selection module.
2. IDEAS300	Variable definition module.
3. IDEAS301	Character listing, validation module.
4. IDEAS302	Application device selection module.
5. IDEASsub	Subroutine module.
6. IDEAS303	Device selections for T, VP, MVP.
7. IDEAS304	Additional device selections for VP, MVP.
8. IDEAS305	Internal device selection module.
9. IDEAS306	Subroutine module to create files or take limits on files.
10. IDEAS307	System date module.
11. IDEAS309	Main menu selection module.

II. IDEAS SYSTEM UTILITIES

A. DATA FILE UTILITIES

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS31M	Menu	ideas31m
2. IDEAS310	Create/revise/re-initilize module	ideas308, ideas 312
3. IDEAS311	Record field definition module	ideas311
4. IDEAS312	Key field selection module	ideas310
5. IDEAS313	Performance option selection module	ideas313
6. IDEAS314	File initialization module	ideas314
7. IDEAS315	Documentation module	ideas31d, ideas315

B. SCREEN MASK UTILITIES

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS32M	Menu	ideas32m
2. IDEAS320	Creation module	ideas308
3. IDEAS321	Revision module	ideas308
4. IDEAS322	Documentation module	ideas308
5. IDEAS323	Printing module	
6. IDEAS32X	Execution	ideas32X

C. REPORT/FORM PRINTING UTILITIES

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS33M	Menu	ideas33m
2. IDEAS330	Creation/revision module	ideas308
3. IDEAS331	Report generator module	ideas331
4. IDEAS332	Execution module	ideas333, ideas331
5. IDEAS333	Execution module	
6. IDEAS334	Documentation module	
7. IDEAS33X	Execution module	

D. APLICATION INITIALIZATION PROGRAM GENERATION

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS35M	Menu	ideas35m
2. IDEAS350	Creation module	ideas308, ideas350
3. IDEAS351	Revision module	ideas308, ideas350
4. IDEAS352	Definition module	ideas350, ideas351 ideas31d

E. APPLICATION MENU PROGRAM UTILITIES

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS36M	Menu	ideas36m
2. IDEAS360	Menu screen & program revision mod.	ideas360
3. IDEAS361	Menu screen & program creation mod.	
4. IDEAS362	Menu screen & program revision mod.	
5. IDEAS363	Documentation program revision mod.	

F. DATA ENTRY/INQUIRY/UPDATE PROGRAM GENERATION

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS37M	Program generation	ideas370
2. IDEAS37X	Module 1 of generated program	
3. IDPROG01	Inquiry module	
4. IDPROG02	Add module	
5. IDPROG03	Add/modify module	
6. IDPROG04	Add/delete module	
7. IDPROG05	Add/modify/delete module	
8. IDPROG06	Modify module	
9. IDPROG07	Delete module	
10. IDPROG08	Modify/delete module	

G. APPLICATION MODULE SELECTION

PROGRAM	DESCRIPTION	ASS. SCREEN (IF ANY)
1. IDEAS30A	Execution of start module	ideas308

APPENDIX E
IDEAS RUN-TIME UTILITIES

The following is a list of program and data files which reside on the IDEAS Application Utilities diskette. Next to each program name are listed the 2200 Hardware configurations which require the program or data file be resident during IDEAS - based application run time.

PROGRAM or DATA FILE	HARDWARE CONFIGURATIONS
ID-SUB-M	MVP
ID-SUB-T	T
ID-SUB-V	VP
IDEAS301	T / VP / MVP
IDEAS306	T / VP / MVP
IDEAS307	T / VP / MVP
IDEAS332	T / VP / MVP
IDEAS333	T / VP / MVP
IDEAS356	T / VP / MVP
IDEAS357	T / VP / MVP
IDEAS358	T / VP / MVP
IDEAS35X	T / VP / MVP
IDEASVAR	MVP
IDFU-300	T / VP / MVP
IDFU-301	VP / MVP
IDFU-302	T / VP/ MVP
IDFU-303	VP / MVP
IDFU-304	VP / MVP
IDFU-305	VP / MVP
IDFU-306	VP/ MVP
IDFU-310	T
IDFU-310	T
IDFU-311	T
IDFU-312	T
IDFU-314	T
IDFU-315	T
IDFU-3M0	T / VP / MVP
IDGBSL-2	MVP
IDGLBSEL	MVP
IDNONGBL	T / VP
IDPRNTSL	T / VP / MVP
ideas307	T / VP / MVP
ideas308	T / VP / MVP
ideas315	T / VP / MVP

ideas31d	T / VP / MVP
ideas331	T / VP / MVP
ideas333	T / VP / MVP
ideas350	T / VP / MVP
idfu-302	T / VP / MVP
idfu-303	T / VP / MVP
idfu-304	T / VP / MVP
idfu-3m0	T / VP / MVP
idfu-3m1	T

APPENDIX F
PRIME NUMBERS

2	3	5	7	11	13	17	19	23
29	31	37	41	43	47	53	59	61
67	71	73	79	83	89	97	101	103
107	109	113	127	131	137	139	149	151
157	163	167	173	179	181	191	193	197
199	211	223	227	229	233	239	241	251
257	263	269	271	277	281	283	293	307
311	313	317	331	337	347	349	353	359
367	373	379	383	389	397	401	409	419
421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523
541	547	557	563	569	571	577	587	593
599	601	607	613	617	619	631	641	643
647	653	659	661	673	677	683	691	701
709	719	727	733	739	743	751	757	761
769	773	787	797	809	811	821	823	827
829	839	853	857	859	863	877	881	883
887	907	911	919	929	937	941	947	953
967	971	977	983	991	997	1009	1013	10109

To help us to provide you with the best manuals possible, please make your comments and suggestions concerning this publication on the form below. Then detach, fold, tape closed and mail to us. All comments and suggestions become the property of Wang Laboratories, Inc. For a reply, be sure to include your name and address. Your cooperation is appreciated.

700-5778A

TITLE OF MANUAL **IDEAS**
(Inquiry Data Entry Access System) User Manual

COMMENTS:

Fold

Fold

(Please tape, Postal regulations prohibit the use of staples.)

WANG

Fold

FIRST CLASS
PERMIT NO. 16
Lowell, Mass.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

— POSTAGE WILL BE PAID BY —

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Attention: Technical Writing Department

Fold

Cut along dotted line.

United States

Alabama
Birmingham
Mobile

Alaska
Anchorage

Arizona
Phoenix
Tucson

California
Culver City
Fountain Valley
Fresno
Inglewood
Sacramento
San Diego
San Francisco
Santa Clara
Ventura

Colorado
Englewood

Connecticut
New Haven
Stamford
Wethersfield

District of Columbia
Washington

Florida
Miami
Hialeah
Jacksonville
Orlando
Tampa

Georgia
Atlanta
Savannah

Hawaii
Honolulu

Idaho
Idaho Falls

Illinois
Chicago
Morton
Park Ridge
Rock Island
Rosemont

Indiana
Indianapolis
South Bend

Kansas
Overland Park
Wichita

Kentucky
Louisville

Louisiana
Baton Rouge
Metairie

Maryland
Rockville
Towson

Massachusetts
Billerica
Boston
Burlington
Chelmsford
Lawrence
Littleton
Lowell
Tewksbury
Worcester

Michigan
Kentwood
Okemos
Southfield

Minnesota
Eden Prairie

Missouri
Creve Coeur

Nebraska
Omaha

Nevada
Las Vegas
Reno

New Hampshire
Manchester

New Jersey
Toms River
Mountainside
Clifton

New Mexico
Albuquerque

New York
Albany
Buffalo
Fairport
Lake Success
New York City
Syracuse

North Carolina
Charlotte
Greensboro
Raleigh

Ohio
Cincinnati
Cleveland
Middleburg Heights
Toledo
Worthington

Oklahoma
Oklahoma City
Tulsa

Oregon
Eugene
Portland

Pennsylvania
Allentown
Camp Hill
Erie
Philadelphia
Pittsburgh
Wayne

Rhode Island
Cranston

South Carolina
Charleston
Columbia

Tennessee
Chattanooga
Knoxville
Memphis
Nashville

Texas
Austin
Dallas
Houston
San Antonio

Utah
Salt Lake City

Vermont
Montpelier

Virginia
Newport News
Norfolk
Richmond

Washington
Richland
Seattle
Spokane
Tacoma

Wisconsin
Brookfield
Madison
Wauwatosa

International Offices

Australia
Wang Computer Pty., Ltd.
Adelaide, S.A.
Brisbane, Qld.
Canberra, A.C.T.
Darwin N.T.
Perth, W.A.
South Melbourne, Vic 3
Sydney, NSW

Austria
Wang Gesellschaft, m.b.H.
Vienna

Belgium
Wang Europe, S.A.
Brussels
Erpe-Mere

Canada
Wang Laboratories (Canada) Ltd.
Burnaby, B.C.
Calgary, Alberta
Don Mills, Ontario
Edmonton, Alberta
Hamilton, Ontario
Montreal, Quebec
Ottawa, Ontario
Winnipeg, Manitoba

China
Wang Industrial Co., Ltd.
Taipei
Wang Laboratories Ltd.
Taipei

France
Wang France S.A.R.L.
Paris
Bordeaux
Lyon
Marseilles
Nantes
Strasbourg
Toulouse

Great Britain
Wang (U.K.) Ltd.
Richmond
Birmingham
London
Manchester
Northwood Hills

Hong Kong
Wang Pacific Ltd.
Hong Kong

Japan
Wang Computer Ltd.
Tokyo

Netherlands
Wang Nederland B.V.
IJsselstein
Gronigen

New Zealand
Wang Computer Ltd.
Auckland
Wellington

Panama
Wang de Panama (CPEC) S.A.
Panama City

Singapore
Wang Computer (Pte) Ltd.
Singapore

Sweden
Wang Skandinaviska AB
Stockholm
Gothenburg
Malmo

Switzerland
Wang A.G.
Zurich
Basel
Geneva

Wang Trading A.G.
Zug

United States
Wang International Trade, Inc.
Lowell, Mass.

West Germany
Wang Laboratories, GmbH
Frankfurt
Berlin
Cologne
Dusseldorf
Essen
Freiburg
Hamburg
Hannover
Kassel
Munich
Nurnberg
Saarbrucken
Stuttgart

International Representatives

Abu-Dhabi
Argentina
Bahrain
Bolivia
Brazil
Canary Islands
Chile
Colombia
Costa Rica
Cyprus
Denmark
Dominican Republic
Ecuador
Egypt
El Salvador
Finland
Ghana
Greece
Guatemala
Haiti
Honduras
Iceland
India
Indonesia
Ireland
Israel
Italy
Jamaica
Japan
Jordan
Kenya
Korea
Kuwait
Lebanon
Liberia
Malaysia
Malta
Mexico
Morocco
Nicaragua
Nigeria
Norway
Paraguay
Peru
Phillippines
Portugal
Saudi Arabia
Scotland
Spain
Sri Lanka
Sudan
Syria
Thailand
Turkey
United Arab Emirates
Venezuela

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

Printed in U.S.A.
700-5778A
4-80-1M